Titulo: Intelligent Advertising

Volumen: 1/1
Alumno: Edilfredo Eliot Díaz Pinedo

Director/Ponente: Fatos Xhafa
Departament: Lenguajes y Sistemas Informáticos
Data: 26 de junio de 2012

DATOS DEL PROYECTO

Titulo del Proyecto: Intelligent Advertising

Nombre del estudiante: Edilfredo Eliot Díaz Pinedo
Titulación: Ingeniería en Informática
Créditos: 37,5
Director/Ponente: Fatos Xhafa
Departamento: Lenguajes y Sistemas Informáticos

MIEMBROS DEL TRIBUNAL *(nombre y firma)*

Presidente: Xavier Molinero Albareda

Vocal: José Ramón Herrero Zaragoza

Secretario: Fatos Xhafa

CALIFICACIÓN

Calificación numérica:
Calificación descriptiva:

Fecha: 26 de junio de 2012

Índice

Índice de figuras

1. Introducción

1.1 Motivación y contexto

La publicidad es el uso de los medios disponibles para persuadir a los clientes de comprar sus productos. Actualmente la publicidad online es el medio de publicidad que más rápido ha crecido desde la aparición de internet, en gran medida porque esta se puede rastrear no solo cuanto tiempo se ha permanecido en la página web con publicidad sino que también cuantas veces alguien hizo clic en la publicidad.

Sin embargo, con los años aparecen nuevas tecnologías y en consecuencia nuevas formas de poder llegar a los potenciales clientes. Este es el caso de los Smartphones (Teléfonos inteligentes) que siendo usado ampliamente por la mayoría de personas permite establecer una nueva forma de publicidad. De acuerdo al Interactive Advertising Bureau (IAB), la inversión en Mobile Advertising (Publicidad móvil) en España en el 2011 fue de 14,32 millones de euros, siendo reconocido el valor de la publicidad en estos dispositivos.

Lo que es realmente importante respecto a los smartphones, además de que constituir un mercado potencialmente grande para la publicidad, son las tecnologías que estas ofrecen y que además de resultar de utilidad a los usuarios pueden ser aprovechadas para mejorar la publicidad móvil. Un ejemplo de ello son las opciones de conectividad por Wi-Fi y 3G, lo cual significa que se pueden llegar a ellos virtualmente en todas partes. Además, la gran mayoría posee capacidad GPS que permite crear un nuevo tipo de publicidad enfoca en la localización (location-based).

El uso de publicidad personalizada que no solo ofrezca productos y servicios que se adecuen a su perfil sino que estos se encuentren a su alrededor, puede ofrecer sin duda una mejor oportunidad de éxito de la publicidad.

1.2 Definición del problema

Actualmente el mayor reto de un negocio es poder atraer clientes y retenerlos. Dada la gran cantidad de tiendas que ofrecen productos del mismo tipo, estas buscan diferenciarse de las demás tratando de destacar sobre las otras. Uno de los mecanismos más empleados para este propósito es el uso de la publicidad.

El uso de la publicidad ayuda a un negocio a hacerse notar por potenciales clientes pero tiene varias problemáticas. Una situación en concreto es la que pasa en un centro comercial, donde a pesar de que estos pueden ser considerablemente grandes, todas las tiendas se encuentran espacialmente cercanas entre ellas ofreciendo productos o servicios similares.

En esta situación, las tiendas que no usan métodos publicitarios aprovechando las nuevas tecnologías, únicamente pueden confiar en que el cliente se sienta atraído por su escaparate o por su marca (son raras las ocasiones donde una persona tenga preferencia por una marca sin haber recibido anteriormente publicidad sobre ella). Esto sin duda pone en gran desventaja a este tipo de comercios pues son menos conocidos y por tanto el cliente puede ignorarla e "ir a las de siempre", perjudicando a la tienda por no atraer clientes y al cliente por perderse la oportunidad de aprovechar alguna promoción publicitaria que le seria ventajosa.

Por otro lado, las tiendas que sí usan publicidades normales tampoco tienen el éxito asegurado. La efectividad de la publicidad empleada por las tiendas puede ser diluida por efecto temporal dado que desde que recibe la publicidad hasta que llega al centro comercial el potencial cliente podría distraerse una vez dentro mientras mira en otras tiendas y así perderse la promoción de la publicidad. Otra vez, la tienda no atrae o retiene clientes y el cliente no logra aprovechar promociones ventajosas.

1.3 Análisis de soluciones existentes

Para reducir el hecho de una publicidad no surta efecto en algún potencial cliente la tendencia actual es hacer llegar la misma a la mayor cantidad de personas posibles esperando que su público objetivo reaccione ante ella. Es un acto usado por la gran mayoría de agencias de publicidad que se traduce en que todas las personas reciban diariamente publicidad por diferentes medios llegándoles a saturar. Esta saturación crea una visión negativa del concepto de publicidad llegándola a catalogar como SPAM e incluso genera rechazo cuando es muy directa provocando que el potencial cliente sea reacio a seguir recibiendo publicidad por ese canal.

Dado que las antiguas formas de publicidad están desgatadas y gracias a la continua aparición de nuevas tecnologías, muchas empresas ahora centran sus esfuerzos en un nuevo canal para enviar publicidad, la publicidad móvil, la cual aprovecha que en la actualidad casi la mitad de la población española tiene un Smartphone y un 52% de sus usuarios la usan mientras hacen sus compras (comscore, 2012).

Existen principalmente dos métodos por el cual hacen llegar publicidad a los potenciales clientes. El primero es hacer uso de proveedores de publicidad que lo que hacen es incrustar las publicidades en paginas web y aplicaciones de terceros. El segundo método es que son las propias empresas quienes crean sus propias aplicaciones para que el propio cliente se mantenga actualizado con toda la información que esta provee.

A pesar de que el uso de los smartphones abre camino a un nuevo mercado, la publicidad móvil todavía presenta ciertas deficiencias que impiden ser correctamente utilizado en un centro comercial. Los proveedores de publicidad usan una serie de filtros predefinidos que fijan las empresas (país, dispositivo, palabra clave, etcétera) y envían la publicidad a todos los usuarios móviles que encajen con el filtro, siendo el conocimiento sobre estos muy limitado. Además no tienen forma de saber si la publicidad tiene éxito o no. Hacer clic en una publicidad no garantiza que el usuario acudirá a la tienda por tanto el indicador CTR[1] (índice de clics en español) no es fiable en este entorno y por tanto es un aspecto que debe mejorarse.

En el caso de que sea la propia empresa quien tenga una aplicación para mantener informado al cliente tiene sus desventajas de parte del cliente. Un centro comercial puede tener más de 20 tiendas y es casi seguro que el usuario estará poco dispuesto a tener tantas aplicaciones solo para mantenerse actualizado sobre las tiendas. Por lo tanto hay que hacer algo para que el usuario no tenga que consultar todas las aplicaciones de las tiendas en cada momento para ver las promociones existentes y también para dar presencia a aquellas tiendas que aun no se publicitan por el uso de aplicaciones propias.

Por último, muchas soluciones que agrupan publicidades de diferentes tiendas están pensadas para los usuarios que se encuentran en la calle (outdoor) y las reglas de publicidad son diferentes cuando uno se encuentra dentro de un recinto (indoor). No hay mucho tiempo desde que el usuario pasa de una zona del centro comercial a otra y los métodos que usan su posicionamiento para informarle de las promociones más cercanas no tienen buena precisión en distancias cortas.

[1] El índice de clic es el porcentaje de impresiones publicitarias sobre las que un usuario ha hecho clic. Nº de clics / Nº de impresiones x 100.

2. Objetivos del proyecto

2.1 Alcance

El PFC que se presenta (Intelligent Advertisement) diseña e implementa un sistema de publicidad para dispositivos móviles en un centro comercial, donde los clientes reciben publicidad de forma pasiva en sus dispositivos mientras están dentro.

La solución que se plantea utiliza tecnologías de location-services (servicios de localización) donde la posición, calculada como unas coordenadas que permiten saber en que parte del centro comercial se encuentra el cliente (por triangulación Wi-Fi o GPS), es un aspecto importante en el momento de evaluar que publicidad enviarle. Otro aspecto de la solución es que el sistema se retroalimenta de la información que las tiendas envían al sistema sobre sus clientes, permitiendo al sistema usar este conocimiento y aplicar unos algoritmos para enviar a cada usuario la publicidad que mejor se adecue a su perfil.

Una ventaja que se plantea en esta solución es que puede sentar base para otros usos. Puede ser completamente escalable ya que podría aplicarse el mismo sistema a una ciudad entera con todas sus tiendas y sus habitantes caminando por sus calles.

2.2 Objetivos

Este éxito del proyecto depende de cumplir los siguientes objetivos generales.

- Permitir que las personas puedan recibir publicidad de forma pasiva en base a su localización y al tiempo.
- Permitir a las tiendas puedan realizar campañas publicitarias orientadas a perfiles en base al conocimiento que se obtiene de los usuarios.
- Elaborar un algoritmo que permita enviar, de forma inteligente, publicidad de las tiendas de un centro comercial a los usuarios.
- Comprender el mercado de la publicidad en la actualidad e investigar nuevas formas de aplicarla en los teléfonos inteligentes.

2.3 Metodología

La metodología que se ha seguido en este proyecto es el Ciclo de vida iterativo o incremental. Este proceso garantiza un alto nivel de éxito en proyectos orientados a objetos.

Este método permite aprovechar la ventaja de aplicar lo que se ha aprendido durante el desarrollo de las anteriores partes del proyecto para mejorarlo. El aprendizaje viene tanto del desarrollo como del uso del sistema, donde los primeros pasos claves en el proceso empiezan con una simple implementación de un subconjunto de los requerimientos del software e iterativamente ir mejorando las continuas versiones hasta que el sistema este completamente implementado. En cada iteración, se hacen modificaciones en el diseño y añaden nuevas capacidades funcionales.

La siguiente figura muestra el proceso descrito anteriormente.

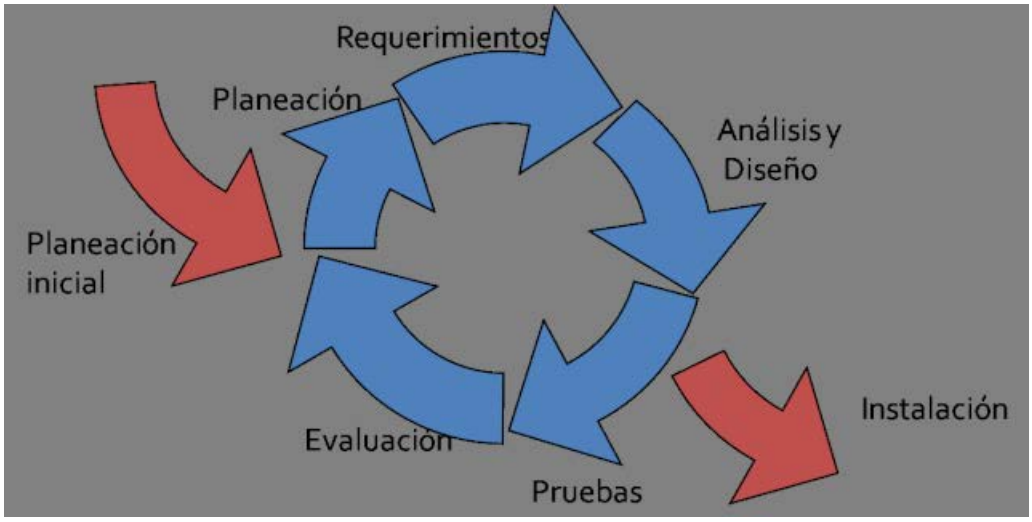


FIGURA 1

2.4 Planificación

En esta sección se explican las diferentes etapas que se han seguido en el desarrollo del proyecto y planificación realizada.

2.4.1 Etapas del proyecto

Definición del proyecto: en reuniones con el director de proyecto se definió el alcance y objetivos del proyecto, además del conjunto de requerimientos

Aprendizaje: periodo de tiempo de familiarización con las tecnologías a utilizar, así como la configuración del entorno. Este periodo es didáctico y se ha realizado durante toda la elaboración del proyecto.

Análisis y especificación: la realización de un análisis de requisitos a partir del cual se realizara su especificación mediante diagramas UML.

Diseño: diseño de la arquitectura y de las especificaciones realizadas al entorno adecuado:

- Diseño capa de presentación
- Diseño capa de negocio
- Diseño capa de datos

Codificación: traducción del modelo conceptual, casos de usos de la especificación y del diseño a componentes de software.

Pruebas: realización de pruebas unitarias y funcionales

Documentación: realización de la documentación del proyecto y de los manuales de usuario, instalación y desarrollo.

2.4.2 Diagrama de Gantt

Nombre de tarea	Duración	Comienzo	Fin
1 Estudio previo	**22 días**	**lun 23/01/12**	**lun 13/02/12**
1.1 Estudio previo de publicidad	7 días	lun 23/01/12	dom 29/01/12
1.2 Estudio de soluciones existentes	8 días	lun 30/01/12	lun 06/02/12
1.3 Aprendizaje de Android	4 días	lun 30/01/12	jue 02/02/12
1.4 Aprendizaje de Maven	4 días	vie 03/02/12	lun 06/02/12
1.5 Aprendizaje Spring	7 días	mar 07/02/12	lun 13/02/12
2 Análisis	**7 días**	**mar 14/02/12**	**lun 20/02/12**
2.1 Análisis de requisitos	3 días	mar 14/02/12	jue 16/02/12
2.2 Análisis funcional	4 días	vie 17/02/12	lun 20/02/12
3 Especificación	**13 días**	**mar 21/02/12**	**dom 04/03/12**
3.1 Modelo casos de uso	13 días	mar 21/02/12	dom 04/03/12
4 Diseño	**32 días**	**jue 08/03/12**	**dom 08/04/12**
4.1 Análisis de tecnologías	4 días	jue 08/03/12	dom 11/03/12
4.2 Capa de datos	7 días	lun 12/03/12	dom 18/03/12
4.3 Capa de negocio	7 días	lun 19/03/12	dom 25/03/12
4.4 Capa de presentación	7 días	lun 26/03/12	dom 01/04/12
4.5 Algoritmo de perfil	8 días	lun 26/03/12	lun 02/04/12
4.6 Algoritmo de publicidad	7 días	lun 02/04/12	dom 08/04/12
5 Implementación	**33 días**	**lun 09/04/12**	**vie 11/05/12**
5.1 Capa de datos	7 días	lun 09/04/12	dom 15/04/12
5.2 Capa de negocio	14 días	lun 16/04/12	dom 29/04/12
5.3 Capa de presentación	5 días	lun 30/04/12	vie 04/05/12
5.4 Algoritmos	7 días	sáb 05/05/12	vie 11/05/12
6 prototipos	**28 días**	**sáb 12/05/12**	**vie 08/06/12**
6.1 Cliente web	7 días	sáb 12/05/12	vie 18/05/12
6.2 Cliente móvil	14 días	sáb 19/05/12	vie 01/06/12
6.3 Servidor	7 días	sáb 02/06/12	vie 08/06/12
7 Pruebas	**64 días**	**lun 16/04/12**	**lun 18/06/12**
7.1 Test unitarios	26 días	lun 16/04/12	vie 11/05/12
7.2 Plan de pruebas	7 días	sáb 09/06/12	vie 15/06/12
7.3 Test funcionales	3 días	sáb 16/06/12	lun 18/06/12
8 Documentación	**132 días**	**mar 14/02/12**	**dom 24/06/12**
8.1 Informe previo	3 días	lun 05/03/12	mié 07/03/12
8.2 Memoria	127 días	mar 14/02/12	mar 19/06/12
8.3 Presentación	5 días	mié 20/06/12	dom 24/06/12

FIGURA 2

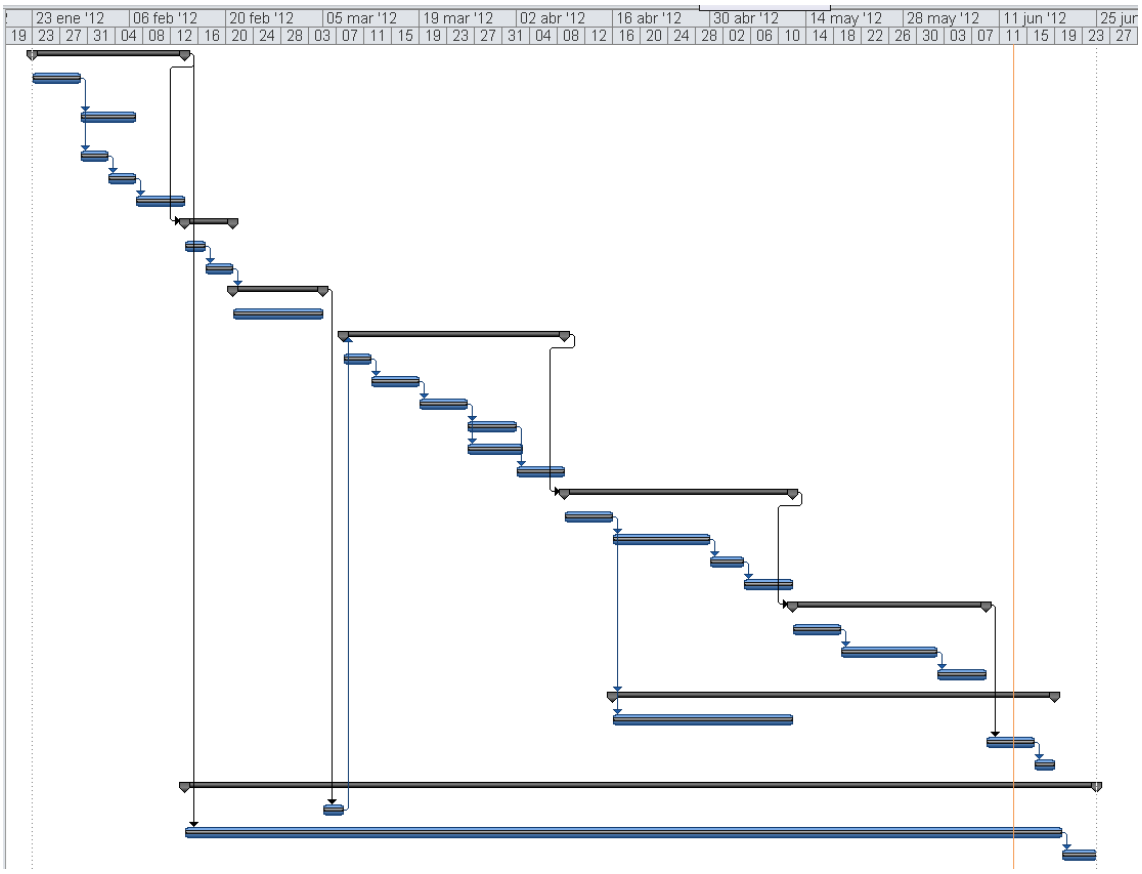

23 ene '12
06 feb '12
20 feb '12
05 mar '12
19 mar '12
02 abr '12
16 abr '12
30 abr '12
14 may '12
28 may '12
11 jun '12
25 jun

FIGURA 3

3. Análisis de requisitos

3.1 Requerimientos funcionales

El sistema debe facilitar el acceso a los usuarios mediante una aplicación móvil. De esta forma podemos distinguir dos tipos de usuarios del sistema.

Tienda. Los usuarios que acceden al sistema a través de la aplicación web.

Cliente. Los usuarios que acceden a la aplicación móvil.

Los usuarios pueden acceder a toda la funcionalidad del sistema. De esta forma el sistema facilitará:

Servicio web. El sistema debe permitir a las tiendas acceder a la información mediante un cliente web.

Servicio móvil. El sistema debe permitir a los clientes acceder a la información a través de un cliente móvil.

Notificación de eventos. Nuestro sistema proporcionara al cliente un mecanismo de notificación de eventos.

Gestión de publicidad. La tienda tiene que ser capaz de crear, modificar y eliminar campañas de publicidad.

Gestión de perfiles. El cliente tiene que ser capaz de crear o modificar la información en su perfil.

Gestión de suscripciones. El cliente tiene que ser capaz de suscribirse y cancelar suscripciones de las tiendas de las cuales quiere recibir publicidad especialmente.

Algoritmo de perfil. El sistema tiene que ser capaz usar la información que obtiene de los clientes para poder generar un perfil de ellos.

Algoritmo de publicidad. El sistema tiene que ser capaz de aprender de la interacción de los clientes con las publicidades enviadas para saber que publicidad enviarles en el futuro.

3.2 Requerimientos no funcionales

Facilidad de uso. Todas las acciones realizables por el sistema no deben exceder de cuatro pasos. El camino tiene que ser el mas intuitivo posible para que el usuario pueda realizar la acción que desea en el camino mas corto y además no habrán dos caminos diferentes desde la pagina de inicio que lleven a realizar la misma acción.

Facilidad de aprendizaje. El sistema necesitara una gran facilidad de aprendizaje, cosa que conseguirá gracias a que se empleen botones de imágenes similares a otros sistemas para que el usuario relacione rápidamente la acción que realiza así como pequeñas ayudas para los campos que indiquen la información que se espera que se introduzca.

Seguridad. Toda información personal de los usuarios será protegida en cumplimiento de la LOPD. Los datos que se recogen desde los móviles sobre perfiles y publicidad no podrán ser facilitadas a otros usuarios que no sean propietarios de los mismo.

Disponibilidad. El sistema tiene que esta disponible durante las 24 horas del día, los 7 días de la semana.

Precisión. Los datos del sistema tienen que ser precisos para evitar incongruencias.

Tiempo de respuesta. El tiempo de respuesta medio es de 0,30 segundos y el máximo permitido no podrá exceder de 4 segundos.

Capacidad. El sistema tiene que soportar la carga de usuarios correspondiente al volumen medio de personas que acuden a un centro comercial en un día. Al menos unos 1000 usuarios que lo usen a la vez.

4. Especificación

En este capitulo se describe el comportamiento que debe tener el sistema de cara al usuario final.

El lenguaje de especificación es UML (Unified Modeling Language), que permite definir modelos previos a la construcción del sistema. La especificación de la ingeniera de software permite definir diferentes modelos en UML, en nuestro caso utilizaremos los siguientes:

Modelo casos de usos: determinar las funciones del sistema para los diferentes actores participantes al sistema.

Modelo de comportamiento del sistema: modelo que nos muestra como interaccionan las entidades externas al sistema y el propio sistema, mediante mensajes que representan funciones del sistema.

4.1 Modelo de casos de usos

El modelo de casos de usos se puede definir a partir de los siguientes elementos fundamentales:

Caso de uso: describe la secuencia de eventos de un agente externo, actor, que utiliza el sistema para realizar procesos que tienen cierto valor para él.

Actor: entidad externa que participa en la historia del caso de uso.

4.1.1 Actores

4.1.1.1 Usuario no registrado

El usuario no registrado tiene como rol principal el descubrir las características principales del sistema. De esta forma, será capaz de ubicarse en el mapa del centro comercial y consultar la lista de tiendas y recibir publicidad únicamente basada en localización y tiempo.

Experiencia y conocimiento del negocio

El usuario no registrado se le considera novel en el conocimiento del negocio ya que puede ser de las primeras veces que accede al sistema y no conoce las ventajas de estar registrado.

Experiencia y conocimiento tecnológico

En cuanto a la experiencia tecnológica esperable, es de suponer que el usuario tenga un nivel aceptable y pueden darse por sentado ciertas convenciones habituales en este ámbito.

Otras características

El usuario no registrado debe ser capaz de encontrarse con una herramienta sencilla en cuanto a uso se refiere y potente en cuanto a prestaciones. Debe tenerse en cuenta que el objetivo del usuario registrado debería ser pasar a ser usuario registrado del sistema. Por lo tanto, se le debe ofrecer un conjunto de facilidades suficientemente amplia como para que compruebe los beneficios que supondrían para él el pasar a operar como cliente registrado.

4.1.1.2 Usuario registrado

El usuario registrado tiene como rol principal usar el sistema en su totalidad creándole un perfil definido que le permitirá enviar publicidad de forma inteligente al usuario y que además el usuario pueda usar los descuentos que se le ofrecen en las tiendas del centro comercial.

Experiencia y conocimiento del negocio

En este caso el usuario registrado puede ser novel si no ha interactuado ni sabe como funciona el sistema o avanzado si esta familiarizado con todas sus funciones

Experiencia y conocimiento tecnológico

Al igual que para el usuario no registro, se asume un cierto nivel de conocimientos tecnológicos que conlleva poseer un Smartphone.

4.1.1.3 Tienda

Tiene como rol crear las campañas publicitarias a partir de la información de los perfiles de los usuarios y el conocimiento que se tiene de estos.

Experiencia y conocimiento del negocio

Tendrá un conocimiento avanzado del negocio que le permitirá llevar a cabo su rol sin ambigüedades.

Experiencia y conocimiento tecnológico

Se espera un conocimiento alto, lo suficiente para poder usar las herramientas que se le provee para sus campañas publicitarias.

4.1.2 Diagrama de casos de uso

Muestra todos los casos de uso del sistema con los actores que interaccionan con él, indicando cuales son las relaciones entre los actores y los casos de uso.

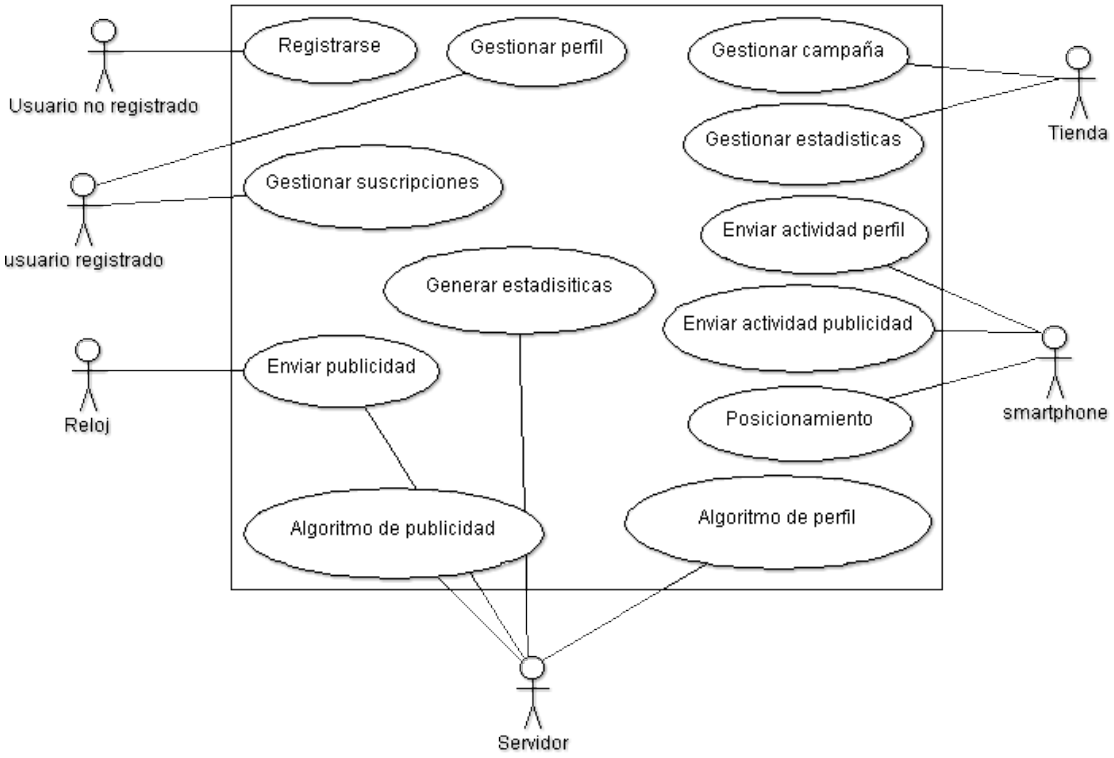


FIGURA 4

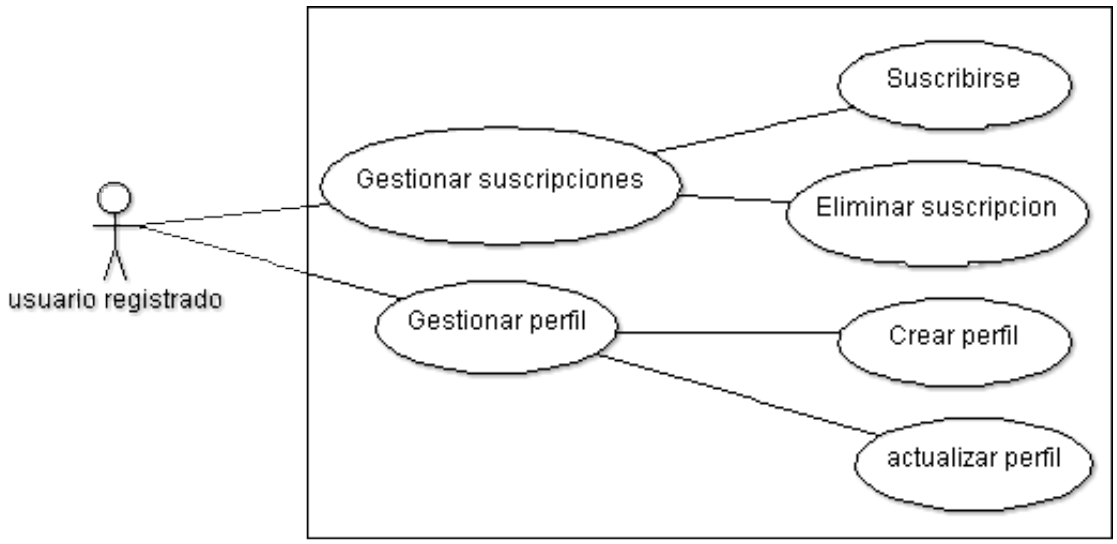


FIGURA 5

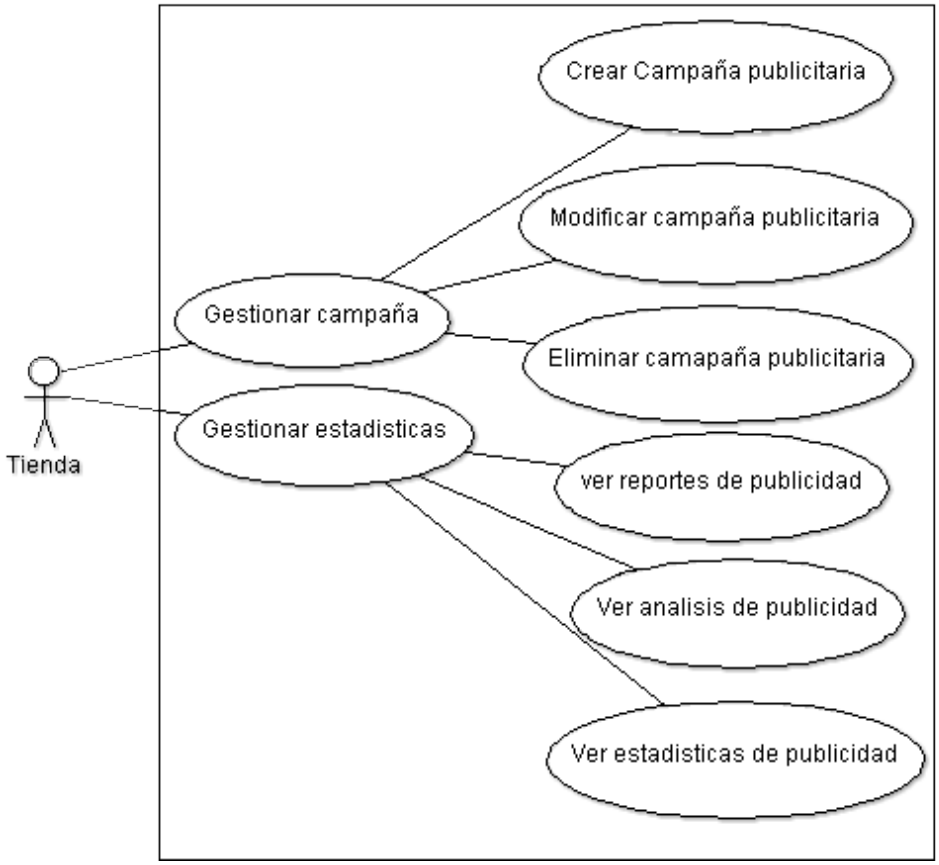

Crear Campaña publicitaria
Modificar campaña publicitaria
Gestionar campaña
Eliminar camapaña publicitaria
Tienda
Gestionar estadisticas
ver reportes de publicidad
Ver analisis de publicidad
Ver estadisticas de publicidad

FIGURA 6

4.1.3 Lista de casos de uso

4.1.3.1 Usuario no registrado

<table>
<tr><td>Caso de uso</td><td>Registrarse</td></tr>
<tr><td>Actores</td><td>Usuario no registrado</td></tr>
<tr><td>Descripción</td><td>Este caso de uso tiene lugar cuando un usuario no esta registrado en el sistema y quiere hacerlo.</td></tr>
<tr><td>Precondiciones</td><td>El usuario no esta registrado en el sistema</td></tr>
<tr><td colspan="2">Flujo básico</td></tr>
<tr><td colspan="2">
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. El usuario quiere registrarse</td><td>2. El sistema muestra al actor la sección de registro</td></tr>
<tr><td>3. El usuario introduce los datos de registro</td><td>4.Se validan los datos y registra al usuario</td></tr>
</table>
</td></tr>
<tr><td>Flujos alternativos</td><td></td></tr>
<tr><td colspan="2">Datos erróneos
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>4. Se indica al usuario que los datos son incorrectos.</td></tr>
</table>
Usuario existente
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>4. Se indica al usuario que ya existe una cuenta asociada a los datos suministrados</td></tr>
</table>
</td></tr>
<tr><td>Postcondiciones</td><td>El usuario esta registrado</td></tr>
</table>

4.1.3.2 Usuario registrado

<table>
<tr><td>Caso de uso</td><td>Iniciar sesión</td></tr>
<tr><td>Actores</td><td>Usuario registrado</td></tr>
<tr><td>Descripción</td><td>Este caso de uso ocurre cuando el usuario quiere iniciar sesión en el sistema con una cuenta que posee.</td></tr>
<tr><td>Precondiciones</td><td>El usuario esta registrado en el sistema</td></tr>
<tr><td colspan="2">Flujo básico</td></tr>
<tr><td colspan="2">
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. El usuario introduce los datos de inicio de sesión</td><td>2. El sistema valida los datos e inicia su sesión</td></tr>
</table>
</td></tr>
<tr><td>Flujos alternativos</td><td></td></tr>
<tr><td colspan="2">Datos erróneos
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>2. Se indica al usuario que los datos son incorrectos.</td></tr>
</table>
Usuario no existente
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>2. Se indica al usuario que no existe una cuenta asociada a los datos suministrados</td></tr>
</table>
</td></tr>
<tr><td>Postcondiciones</td><td>El usuario ha iniciado sesión</td></tr>
</table>

Caso de uso	**Actualizar perfil**
Actores	Usuario ha iniciado sesión
Descripción	Este caso de uso tiene lugar cuando el usuario quiere ajustar sus gustos, preferencias y suscripciones a tiendas.
Precondiciones	El usuario ha iniciado sesión en el sistema
Flujo básico	

Actor	Sistema
1. El usuario selecciona la opción de palabras claves.	2. Se muestran todas las palabras claves introducidas por el usuario en su perfil
3. Se introduce las nuevas palabras claves del usuario	4. Se actualizan las palabras claves del usuario en el sistema y recalcula la relevancia de todas sus palabras claves.

Flujos alternativos	

Opción suscripciones

Actor	Sistema
1. El usuario selecciona la opción de suscripciones	2. Se muestran todas las tiendas a la cual el usuario esta suscrito.
3. selecciona aquellas tiendas en las que quiere cancelar la suscripción.	4. Se eliminan las tiendas seleccionadas de las suscripciones del usuario.

Postcondiciones	El usuario ha modificado su perfil personal

Caso de uso	**Ver mapa**
Actores	Usuario registrado / usuario no registrado
Descripción	Este caso de uso tiene lugar cuando el usuario quiere ver el mapa del centro comercial con la ubicación de las tiendas y situar su posición dentro del recinto.
Precondiciones	-
Flujo básico	

Actor	Sistema
	1. El sistema triangula las coordenadas de la posición del cliente y carga el mapa de centro comercial y se lo muestra por pantalla mostrándole su posición en la que se encuentra él en ese momento

Flujos alternativos	

Buscar tienda

Actor	Sistema
2. Se introduce el nombre de la tienda que busca.	3. El sistema obtiene las coordenadas de las tienda y le muestra la ruta entre la tienda y el usuario.

El usuario no tiene el localizador activado

Actor	Sistema
	1. El sistema carga el mapa del centro comercial y lo muestra por pantalla.

Postcondiciones	El usuario ha visto el mapa

Caso de uso	**Ver lista publicidad**
Actores	Usuario registrado
Descripción	Este caso de uso tiene lugar cuando el usuario quiere ver todas las publicidades que le han sido enviadas.
Precondiciones	El usuario ha iniciado sesión en el sistema
Flujo básico	

Actor	Sistema
	1. Se muestra la lista de publicidades que ha recibido agrupada por categorías.

Flujos alternativos	
Postcondiciones	El usuario ha visualizado la lista de publicidades

Caso de uso	**Suscribirse**
Actores	Usuario registrado
Descripción	Este caso de uso tiene lugar cuando el usuario ha seleccionado una tienda y quiere suscribirse para recibir todas las noticias y promociones que esta ofrezca.
Precondiciones	El usuario ha iniciado sesión en el sistema.
Flujo básico	

Actor	Sistema
	1. El sistema suscribe al cliente a la tienda.

Flujos alternativos	
Postcondiciones	El usuario se suscribe a una tienda

Caso de uso	**Eliminar suscripción**
Actores	Usuario registrado
Descripción	Este caso de uso tiene lugar cuando el usuario quiere cancelar una suscripción que tenia con una tienda.
Precondiciones	El usuario ha iniciado sesión en el sistema. El usuario está suscrito a alguna tienda.
Flujo básico	

Actor	Sistema
1. El usuario indica la tienda que quiere cancelar la suscripción.	2. El sistema cancela la suscripción y elimina todas publicidades asociadas a esta tienda de su lista de publicidad.

Flujos alternativos	
Postcondiciones	El usuario ha eliminado su suscripción.

Caso de uso	**Ver lista tiendas**
Actores	Usuario registrado / usuario no registrado
Descripción	Este caso de uso tiene lugar cuando el usuario quiere ver la lista de tiendas del centro comercial con la información asociada a cada una.
Precondiciones	-
Flujo básico	

Actor	Sistema
	1. Se muestra la lista de tiendas del centro comercial ordenada por categorías

Flujos alternativos	

Ver tienda

Actor	Sistema
2. Selecciona una tienda de la lista	3. Se muestra toda la información que se dispone de la tienda seleccionada.

Postcondiciones	El usuario ha visto la lista de tiendas

Caso de uso	**Ver código**
Actores	Usuario registrado
Descripción	Este caso de uso tiene lugar cuando el usuario quiere ver el código único asociado a la publicidad que ha recibido.
Precondiciones	El usuario ha iniciado sesión en el sistema. El usuario ha recibido alguna publicidad.
Flujo básico	

Actor	Sistema
1. Selecciona la opción ver código de la publicidad	2. Se genera un código único para ese cliente y publicidad y se lo envía al cliente. Además guarda el código asociado del cliente y publicidad.

Flujos alternativos	
Postcondiciones	El usuario recibe un código de la publicidad

4.1.3.3 Tienda

<table>
<tr><td>Caso de uso</td><td>Crear publicidad</td></tr>
<tr><td>Actores</td><td>Tienda</td></tr>
<tr><td>Descripción</td><td>Este caso de uso tiene lugar cuando la tienda quiere crear una nueva campaña publicitaria para los usuarios del sistema.</td></tr>
<tr><td>Precondiciones</td><td>-</td></tr>
<tr><td colspan="2">Flujo básico</td></tr>
<tr><td colspan="2">
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>1. Se pide la información básica necesaria para crear la campaña publicitaria e imágenes que desee utilizar.</td></tr>
<tr><td>2. Se introducen todos los datos necesarios y configura la presentación visual de la misma.</td><td>3. Se guarda la publicidad y solicita los atributos que deben cumplir los perfiles de los clientes a quienes va dirigida la publicidad.</td></tr>
<tr><td>4. Se introducen las palabras claves de la publicidad, tipo de perfil de cliente asociado, franja horaria que desea que se active y otras características.</td><td>5. Se guardan las opciones de la publicidad y se guarda como pendiente de activación.</td></tr>
</table>
</td></tr>
<tr><td>Flujos alternativos</td><td></td></tr>
<tr><td colspan="2">Datos erróneos
<table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>3. Indica los datos introducidos que son incorrectos y debe corregir. [Ir a 2]</td></tr>
</table>
</td></tr>
<tr><td>Postcondiciones</td><td>El usuario ha creado una publicidad sin activar.</td></tr>
</table>

Caso de uso	**Activar publicidad**
Actores	Tienda
Descripción	Este caso de uso tiene lugar cuando la tienda desea activar la campaña publicitaria creada y pueda ser enviada a los usuarios que encajen con el perfil elegido.
Precondiciones	La tienda tiene alguna campaña sin activar
Flujo básico	

Actor	Sistema
1. Selecciona la campaña publicitaria que quiere activar.	2. El sistema pide las propiedades de activación que debe cumplir la publicidad.
3 El actor el rango de fecha durante el cual estará activo	4. El sistema esta listo para enviar la publicidad cuando encuentre un cliente cuyo perfil se ajuste a la publicidad.

Flujos alternativos	
Postcondiciones	La publicidad se ha activado

Caso de uso	**Desactivar publicidad**
Actores	Tienda
Descripción	Este caso de uso tiene lugar cuando la tienda desea desactivar una campaña publicitaria para que deje de ser enviada a los usuarios.
Precondiciones	La tienda tiene alguna campaña activada
Flujo básico	

Actor	Sistema
1. Selecciona la campaña publicitaria que quiere desactivar.	2. El sistema pide confirmación.
3. Confirma la desactivación	4. El sistema marca la publicidad como desactivada y deja de enviarla a los usuarios.

Flujos alternativos	
Postcondiciones	La publicidad se ha desactivado

Caso de uso	**Validar código**
Actores	Tienda
Descripción	Este caso de uso tiene lugar cuando la tienda valida el código de oferta que recibió un usuario para poder aplicarle el descuento.
Precondiciones	-
Flujo básico	

Actor	Sistema
1. Introduce el código.	2. El sistema valida el código y registra que la publicidad ha sido usada por el cliente móvil.

Flujos alternativos	
Postcondiciones	El código ha sido validado y la oferta se aplica

4.1.3.4 Smartphone

Caso de uso	**Enviar actividad perfil**
Actores	Smartphone
Descripción	Este caso de uso tiene lugar cuando el Smartphone necesita notificar de algún cambio o nuevos datos que se introducen en el perfil de usuario (pe: suscripciones, tags, etcétera)
Precondiciones	El usuario ha iniciado sesión.
Flujo básico	

Actor	Sistema
1. se envían los nuevos datos o los que se modifican.	1. El sistema almacena los nuevos datos para redefinir su perfil en la siguiente ejecución del algoritmo de perfil.

Flujos alternativos	
Postcondiciones	El Smartphone ha enviado al sistema la actividad del perfil

<table>
<tr><td>Caso de uso</td><td>Enviar actividad publicidad</td></tr>
<tr><td>Actores</td><td>Smartphone</td></tr>
<tr><td>Descripción</td><td>Este caso de uso tiene lugar cuando el Smartphone necesita notificar al sistema de las interacciones del usuario con la publicidad.</td></tr>
<tr><td>Precondiciones</td><td>El usuario ha iniciado sesión.</td></tr>
<tr><td colspan="2">Flujo básico</td></tr>
<tr><td colspan="2"><table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td>1. Envía todas las acciones realizadas por el usuario sobre las publicidades en un formato predefinido.</td><td>1. El sistema almacena los nuevos datos para redefinir su perfil en la siguiente ejecución del algoritmo de perfil y para ayudar al sistema a identificar las publicidades relevantes para el usuario en la próxima ejecución del algoritmo de publicidad.</td></tr>
</table></td></tr>
<tr><td>Flujos alternativos</td><td></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Postcondiciones</td><td>El Smartphone ha enviado al sistema la actividad de la publicidad.</td></tr>
</table>

Caso de uso	**Posicionamiento**
Actores	Smartphone
Descripción	Este caso de uso tiene lugar cuando el Smartphone quiere notificar al sistema de su posición actual en el centro comercial.
Precondiciones	-

Flujo básico

Actor	Sistema
1. Envía las coordenadas GPS actuales	2. El sistema almacena la posición actual del Smartphone dentro del centro comercial permitiéndole saber si se encuentra dentro de una tienda o no y en zona se encuentra.

Flujos alternativos	

Cambio de zona

Actor	Sistema
	3. detecta que ha cambiado de zona. Ir a "Enviar publicidad"

GPS no disponible

Actor	Sistema
1. triangula su posición usando los AP mediante WIFI y envía las coordenadas estimadas al sistema.	2. El sistema almacena la posición actual del Smartphone.

Postcondiciones	El sistema sabe la ubicación del usuario dentro del centro comercial

4.1.3.5 Sistema

Caso de uso	**Enviar publicidad**
Actores	Sistema
Descripción	Este caso de uso tiene lugar cuando el sistema se dispone a enviar publicidad ya sea activado por una franja horaria o por la localización del usuario
Precondiciones	-
Flujo básico	

Actor	Sistema
	1. Obtiene los usuarios elegibles para enviarles publicidad y les envía aquella que el algoritmo de perfil y publicidad evalúa es adecuada.

Flujos alternativos	
Postcondiciones	El sistema ha enviado publicidad a todos los usuarios que el algoritmo ha seleccionado.

Caso de uso	**Algoritmo de perfil**
Actores	Sistema
Descripción	Este caso de uso tiene lugar cuando el sistema se dispone a ejecutar el algoritmo de perfil para crear los perfiles de nuevos usuarios y redefinir los ya existentes después de un periodo de tiempo.
Precondiciones	-
Flujo básico	

Actor	Sistema
	1. Se ejecuta el algoritmo y redefine el perfil de los clientes móviles.

Flujos alternativos	
Postcondiciones	El sistema ejecutado el algoritmo de perfil

<table>
<tr><td>Caso de uso</td><td>Algoritmo de publicidad</td></tr>
<tr><td>Actores</td><td>Sistema</td></tr>
<tr><td>Descripción</td><td>Este caso de uso tiene lugar cuando el sistema se dispone a ejecutar el algoritmo de publicidad para establecer las publicidades prioritarias de cada perfil después de un periodo de tiempo.</td></tr>
<tr><td>Precondiciones</td><td>-</td></tr>
<tr><td colspan="2">Flujo básico</td></tr>
<tr><td colspan="2"><table>
<tr><td>Actor</td><td>Sistema</td></tr>
<tr><td></td><td>1. Se ejecuta el algoritmo y establece las publicidades que deben ir asociadas a cada perfil.</td></tr>
</table></td></tr>
<tr><td>Flujos alternativos</td><td></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Postcondiciones</td><td>El sistema ejecutado el algoritmo de publicidad.</td></tr>
</table>

5 Diseño

La fase siguiente a la especificación, en un proyecto de desarrollo *software*, es la fase de diseño, donde se va a estudiar y decidir cómo va a funcionar el sistema, basándonos en su arquitectura, siempre considerando todos los requerimientos no funcionales, con el propósito de definir un sistema con el suficiente detalle como para permitir su construcción física (implementación).

Los objetivos principales del diseño son:

- Construir el sistema en base al lenguaje de programación utilizado y/o plataforma.
- Conocer y analizar en profundidad los requerimientos no funcionales del sistema para poder así definir la relación entre cada uno de los componentes que forman el sistema y obtener una arquitectura de calidad acorde a los requerimientos establecidos.
- Descomponer la fase de implementación en varias partes facilitando así su posible gestión en equipos de desarrollo diferentes.
- Crear una abstracción del desarrollo de todos los componentes del sistema. Esto permitirá en un futuro la posibilidad de reingeniería inversa entre el diseño del sistema y su implementación.

En primer lugar se procederá a describir los patrones arquitectónicos junto a la arquitectura física de los sistemas para que éstos se adecuen de la mejor forma posible a las funcionalidades especificadas en el capítulo anterior. Finalmente se realizará una descripción detallada de la arquitectura lógica del sistema global, de tal forma que se puedan mostrar los diferentes componentes *software* existentes en el sistema.

5.1 Patrones arquitectónicos

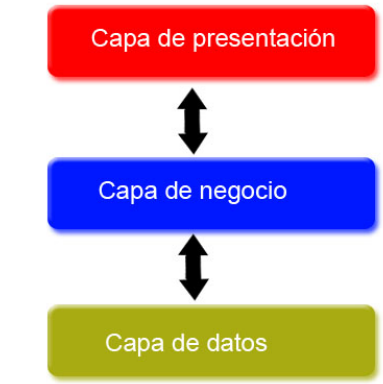
El primer paso del diseño consiste en diseñar la arquitectura del sistema a implementar. Los patrones arquitectónicos son patrones de diseño que ayudan a fijar la estructura global de la aplicación.

Existen varios tipos de patrones arquitectónicos, cada uno dirigido a ofrecer soluciones a diferentes problemas de arquitectura de software. En concreto se han elegido el patrón Modelo – Vista – Controlador para la interfaz web y la arquitectura por capas para la interfaz móvil. A continuación se presenta la justificación de usar ambos patrones.

5.1.1 Patrón de programación por capas

Es una especialización del patrón cliente-servidor donde la carga se divide en tres partes (o capas) con un reparto claro de funciones:

- Capa de presentación, donde se encuentra la interfaz de usuario.
- Capa de negocio, donde se realizan los cálculos y se encuentra modelado el negocio.
- Capa de datos, que permite mantener la persistencia.



Arquitectura en 3 capas

FIGURA 7

Los principales beneficios del estilo de arquitectura de N-capas/3-capas son:

- **Mejoras en las posibilidades de mantenimiento**. Debido a que cada capa es independiente de la otra los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.
- **Escalabilidad**. Como las capas están basadas en diferentes máquinas, el escalamiento de la aplicación hacia afuera es razonablemente sencillo.
- **Flexibilidad**. Como cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa.
- **Disponibilidad**. Las aplicaciones pueden aprovechar la arquitectura modular de los sistemas habilitados usado componentes que escalan fácilmente lo que incrementa la disponibilidad.

5.1.1.1 Capa de presentación

Es la capa que ve el usuario, también se la denomina "capa de usuario", presenta el sistema al usuario, le comunica la información y captura la información del usuario. La interfaz gráfica debe tener la característica de ser entendible y fácil de usar para el usuario. Esta capa se comunica únicamente con la capa de negocio.

En nuestro diseño tenemos dos interfaces en la capa de presentación, el interfaz web y el interfaz móvil.

Interfaz móvil

Es el punto de entrada de los usuarios móviles. Es una aplicación que se ejecuta sobre smartphones con una mínima capacidad de procesamiento.

Con el fin de que la aplicación sea lo más ligera posible, todos los datos dinámicos que se necesiten se facilitan mediante una comunicación con la lógica de negocios.

Interfaz web

Es el punto de entrada a los usuarios web del sistema. Esta interfaz debe facilitar al usuario interactuar con toda la lógica de negocio

5.1.1.2 Capa de negocio

También denominada capa de dominio, es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y se comunica con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

5.1.1.3 Capa de datos

Es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

5.1.2 Patrón Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador (MVC) es otro patrón arquitectónico muy usado, principalmente orientado hacia aplicaciones web, que define en que capas se estructura lógicamente la aplicación (Modelo, Vista y Controlador) pero que además detalla las responsabilidades exactas de cada capa y la forma que tienen que relacionarse entre sí.

La principal diferencia respecto a la programación en tres capas es que la capa Modelo incluiría la capa de negocio y datos, la Vista incluiría el interfaz de usuario (parte de la capa de presentación) y el Controlador seria una mezcla entre la capa de presentación y de negocio.

El patrón MVC se divide en tres módulos:

5.1.2.1 Modelo

Representa los datos y la implementación lógica de la aplicación. Usualmente es responsable de:

- Guardar, borrar, actualizar los datos de la aplicación. Esto por lo general incluye operaciones sobre la base de datos.
- Encapsular la lógica de la aplicación. Esta es la capa que debería implementar toda la lógica de la aplicación. Los errores más comunes son implementar las operaciones lógicas dentro de la capa controlador o vista.

5.1.2.2 Vista

Es la capa de presentación, responsable de dar formato a los datos recibidos del modelo en una forma accesible para el usuario. Los datos pueden venir en diferentes formatos desde el modelo: objetos simples, estructuras XML, JSON, etcétera.

5.1.2.3 Controlador

Es el que recibe las peticiones, las parsea, invoca el modelo para que realice las operaciones solicitadas, toma la respuesta del modelo y se la envía a la capa de presentación. Prácticamente es el vínculo entre la vista y el modelo para que ambos trabajen juntos.

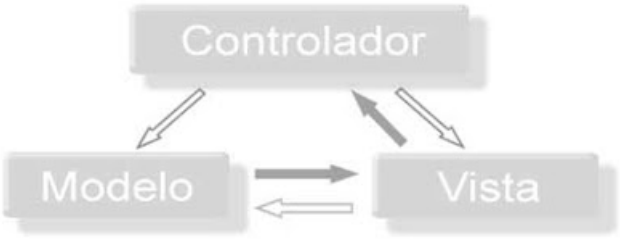


FIGURA 8

Usar este patrón tiene muchas ventajas, las cuales son:

- El modelo y la vista están separados, haciendo la aplicación más flexible.
- El modelo y la vista pueden ser cambiados o remplazados por separado. Por ejemplo una aplicación puede usar servicios web en el Back-End en lugar de una base de datos con solo remplazar el modulo del modelo.
- Cada modulo puede ser probado por separado.

5.2 Arquitectura

El proyecto se compone de dos partes, la parte del cliente y la parte del servidor de publicidad. En la siguiente figura se puede apreciar la visión general de la arquitectura. De esta forma, la arquitectura básica del sistema sigue el patrón en 3 capas, que se justifica en las siguientes razones:

Escalabilidad

Una de las necesidades de este proyecto es que sea escalable, sin que por ello se creen grandes costes en la ampliación del sistema. La posibilidad de aumentar la capacidad de clientes en áreas más grandes y ampliar la red de servidores si que se vean afectos entre ellos es una gran ventaja, ya que la idea de este proyecto es que pueda sentar como base para una implementación a nivel de ciudades.

Centralización

La centralización de datos y existencia de una única lógica de negocio en el servidor evita que exista una replicación de datos en los dispositivos de los clientes. También permitirá fácilmente la realización de cambios y actualizaciones de los datos.

5.2.1 Visión global

El sistema global esta conformado por programas clientes (en este caso tanto la aplicación móvil como la aplicación web), que realizan una serie de peticiones al servidor. Estas peticiones varían según el cliente. En el caso del cliente móvil este se encarga de hacer peticiones al servidor para comprobar si existe alguna publicidad para su perfil. Además dentro de la misma aplicación el cliente puede especificar de que tiendas o productos desea recibir publicidad de entre todos los disponibles en el servidor como también, opcionalmente, establecer una configuración simple de como desea recibirla.

Por otro lado, la tienda, que accede a través de la web, puede introducir las publicidades que desea hacer llegar a los usuarios de la aplicación móvil y recibir reportes estadísticos del éxito que esta teniendo su publicidad.

Para que el sistema pueda generar conocimiento sobre los clientes y generar un perfil para ellos, el sistema debe retroalimentarse (aprender) de acciones pasadas. Es decir, utilizar minería de datos sobre la actividad del cliente durante el pasado para identificar sus gustos y preferencias con máxima precisión. Para lograr esto se utilizan dos algoritmos (de perfil y de publicidad) que definen como extraer los datos y transformarlo en conocimiento.

Por último, para poder medir el éxito de la publicidad de forma real y, aprovechando la ventaja de las nuevas tecnologías en smartphones, se utiliza un sistema de validación por código único por cliente y publicidad y que también es aprovechado de forma importante en los algoritmos.

A continuación se puede observar de forma esquemática como funciona el sistema global:

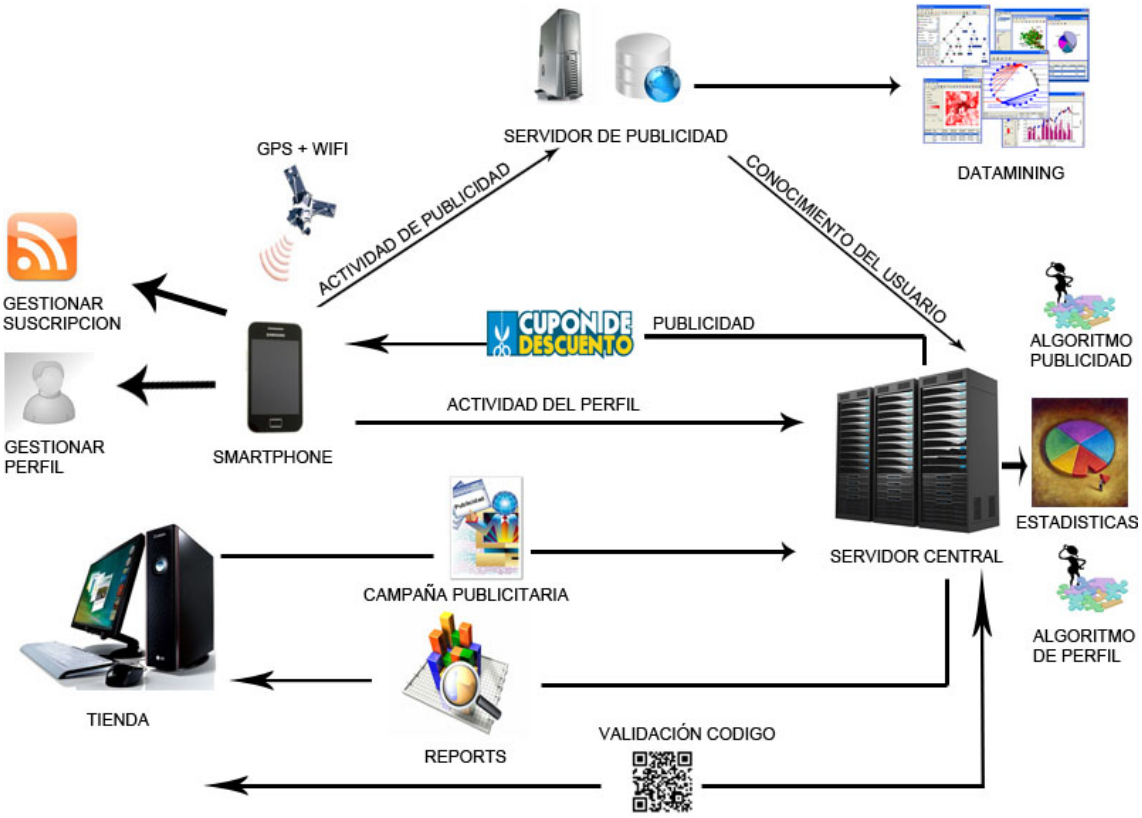


FIGURA 9

5.2.2 Arquitectura física

El cliente móvil es responsable de recoger la posición actual del usuario cada cierto tiempo, usando triangulación Wi-Fi si no esta disponible el GPS, y enviársela al servidor. De esta forma, mientras se detecte que el cliente se encuentre en el centro comercial, se mantendrá un registro histórico de todas las zonas por donde ha pasado y durante cuanto tiempo. El cliente móvil también se encarga de informar de cualquier cambio que se haya realizado del perfil de este y enviar notificaciones de toda la actividad que este usuario haya realizado con la publicidad que haya recibido, por ejemplo, que publicidad ha visto, cuál ha querido pedir el código de promoción, cual ha descartado, etcétera.

Para evitar un excesivo consumo de ancho de banda, el propio cliente móvil almacena en su propia base de datos en la aplicación toda la información que consulta continuamente y no varia con el tiempo. De esta manera evita consulta reiteradas sobre datos redundantes.

Las tiendas son responsables de la comprobación y validación de los códigos promocionales que los clientes móviles le muestra. La tienda se encarga de enviar el código al servidor para que su validación sea efectiva.

Por el lado del servidor, cuando los clientes móviles envían los datos sobre la actividad con la publicidad, estas son enviadas al ***servicio de publicidad*** que filtra los datos recibidos y envía aquellos que considera relevantes al ***Gestor de uso*** que extrae y transforma toda información que obtenga de esta actividad y la guarda en la base de datos.

Cuando el cliente móvil envía las coordenadas de posición actuales al ***servicio de localización***, este evalúa si esta dentro del centro comercial (si están usando el GPS) y solo en tal caso compara su posición con la última que tenía para comprobar su velocidad media de avance o si no se mueve de una posición (pe: se encuentra en un punto de su gran interés o comiendo). Toda esta información se la envía al ***Gestor de mapa*** que la almacena en la base de datos y la evalúa en un escenario donde dispone de todas coordenadas del centro comercial y posiciones de todas las tiendas. Si nota algún cambio relevante (pe: algún cambio de zona en el centro comercial), se lo notifica al ***Gestor de publicidad*** para que tome las medidas necesarias.

El ***servicio de perfil*** recibe todos los cambios que el cliente móvil notifica en su perfil y envía al ***Gestor de conocimiento*** la información inferida de este cambio, el cual utiliza el "algoritmo de perfil" para extraer conocimiento de este cambio y estimar sus nuevos gustos e intereses. Este conocimiento es enviado al ***Gestor de publicidad*** para que la tome las medidas necesarias.

Tanto ***Gestor de uso***, el ***Gestor de mapa***, como el ***Gestor de conocimiento*** utilizan algoritmos de ***Minería de datos*** para obtener un mejor conocimiento del cliente móvil. **El *Gestor de publicidad*** utiliza toda la información enviada del ***Gestor de uso***, el ***Gestor de mapa*** y del ***Gestor de conocimiento*** para ejecutar el "algoritmo de publicidad" e informar al ***Selector de publicidad*** qué tipo de publicidad debe recibir el cliente móvil, cuándo y a qué hora, si cabe.

El ***servicio de código*** se encarga de generar los códigos promocionales asociados a cada publicidad cuando los clientes móviles lo soliciten y de validarlos y registrar su uso cuando las tiendas quieran comprobarlos.

A continuación se puede observar como esta constituido la arquitectura física del sistema:

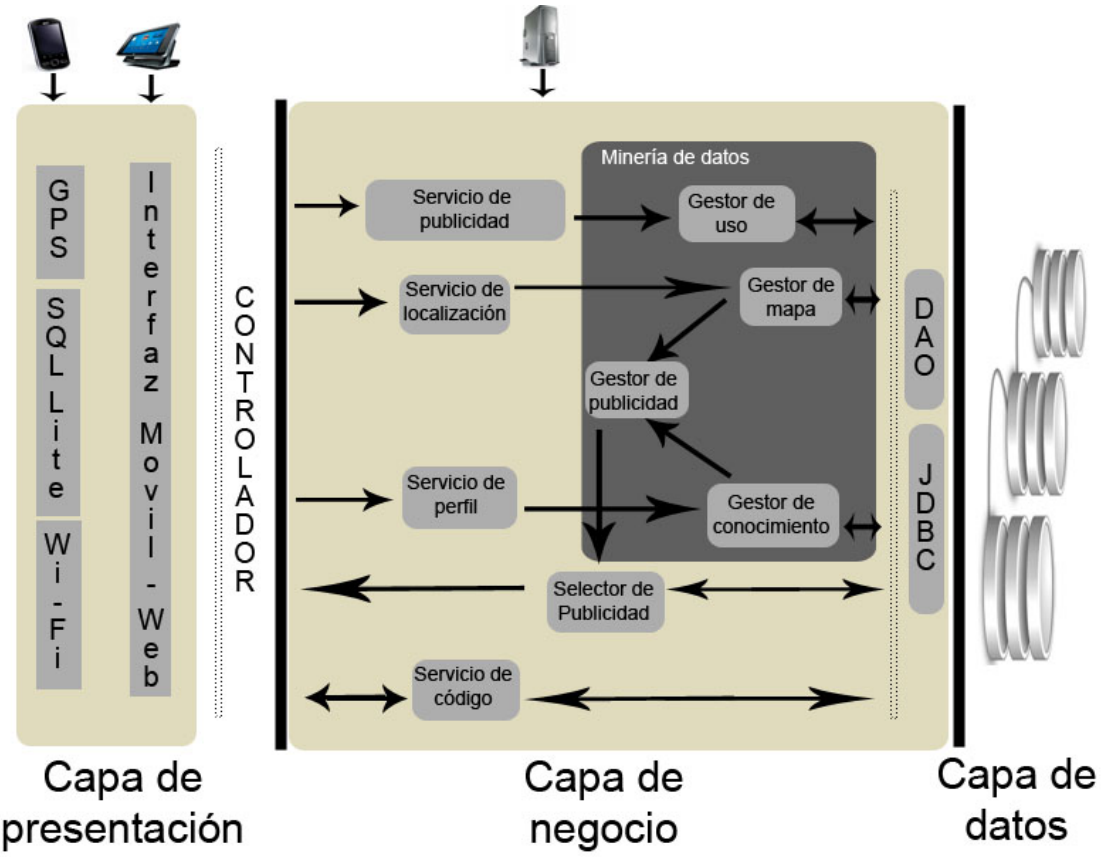


FIGURA 10

5.3 Arquitectura lógica

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaz.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

5.3.1 Capa de presentación

5.3.1.1 Patrón Front Controller

Los sistemas software reciben eventos , por ejemplo, la capa de presentación recibe eventos de la interface de usuario o la capa de negocio recibe eventos externos. Estos eventos interceptados deben ser recibidos por algún objeto del sistema y ejecutar la petición solicitada. En este proyecto, el objeto encargado de recibir los eventos es el controlador.

Este patrón establece un punto inicial de contacto para manejar las peticiones y tratarlas según el tipo que esta sea. El controlador provee un punto de entrada centralizado y gestiona todas las peticiones web o de servicio. Centralizando el control en el controlador y reduciendo la lógica de negocio en la vista permite que el código pueda ser reutilizado entre peticiones.

Las principales ventajas del patrón son:

- Procesamiento de los servicios comunes del sistema por petición. Por ejemplo, el servicio de seguridad realiza la comprobación de autenticación y autorización.
- Permite tener la lógica en una localización central, manejándola mejor en lugar tenerla replicada en diferentes vistas.
- Existen puntos de decisión con respecto a la recuperación y manipulación de datos.
- Se pueden usar múltiples vistas para responder a peticiones de negocio similares.
- Un punto de contacto centralizado para manejar las peticiones puede ser útil para, por ejemplo, controlar y registrar la actividad del usuario a través del sitio.
- Los servicios del sistema y gestión de vistas lógicas son relativamente sofisticados.

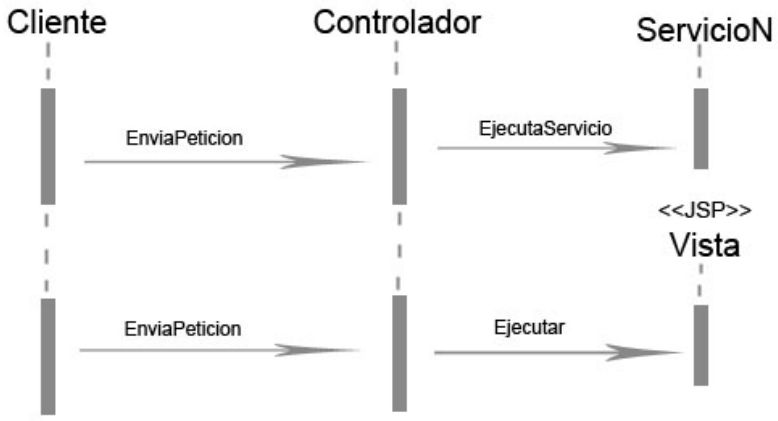


FIGURA 11

5.3.2 Capa de negocio

5.3.2.1 Patrón observador

Los eventos son acciones que suceden durante la ejecución del sistema o mediante la interacción del usuario con el sistema. En el primer caso, el sistema puede lanzar un evento de forma automática como respuesta a un fallo o error interno. Mientras que en el segundo caso podemos indicar al sistema que acciones debe tomar en función de la interacción del usuario. Este proyecto se centrara en el segundo tipo de evento.

El cliente móvil puede generar un evento cuando realiza peticiones a los servicios del sistema y este debe ejecutar los procesos correspondientes que deben tratarlos. Para poder implementar la notificación de eventos dentro del sistema se utilizar el patrón observador, que permite desacoplar las clases de los objetos del sistema, aumentando la modularidad del lenguaje, así como evitar bucles de actualización (espera activa o polling).

Este patrón se conoce también como el patrón de publicación-inscripción, el cual sugiere las ideas básicas del patrón, que son sencillas: el objeto de datos, llamémosle “sujeto”, contiene atributos mediante los cuales cualquier objeto observador o vista se puede suscribir a él pasándole una referencia a sí mismo. El sujeto mantiene así una lista de referencia a sus observadores.

Los observadores a su vez están obligados a implementar unos métodos determinados mediante los cuales el sujeto es capaz de notificar a los observadores “suscritos” los cambios

que sufre para que todos ellos tengan la oportunidad de refrescar el contenido representado. De manera que cuando se produce un cambio en el sujeto, el resto de observadores es notificado de este cambio.

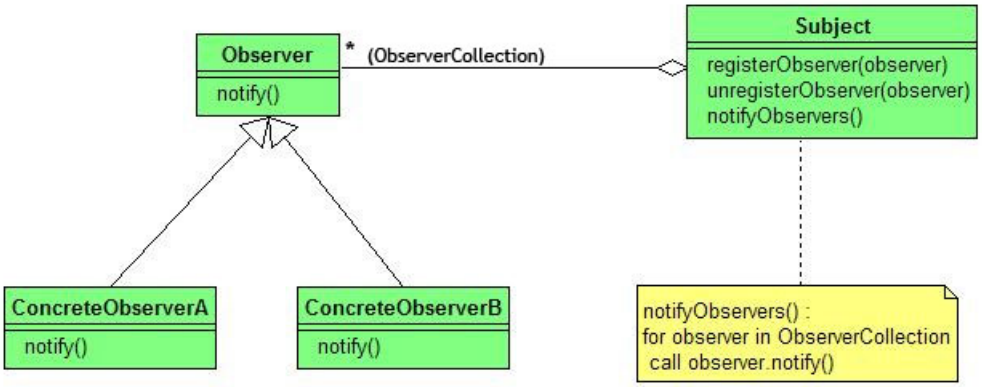


FIGURA 12

5.3.3 Capa de datos

5.3.3.1 Patrón DAO

Los DAO implementan un mecanismo de acceso para trabajar con una fuente de datos determinada. Esta fuente puede ser tan diversa como una base de datos relacional, un servicio externo de datos, ficheros, entre otros. El componente de negocio se ayuda en la interfaz sencilla proporcionada por el DAO, escondiendo completamente los detalles de la implementación. Como esta interfaz no varia cuando cambia la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin afectar a los componentes de negocio.

Esencialmente los DAO actúan como adaptadores entre los componentes y las fuentes de datos.

La siguiente figura muestra el diagrama de clases relacionales con el patrón DAO.

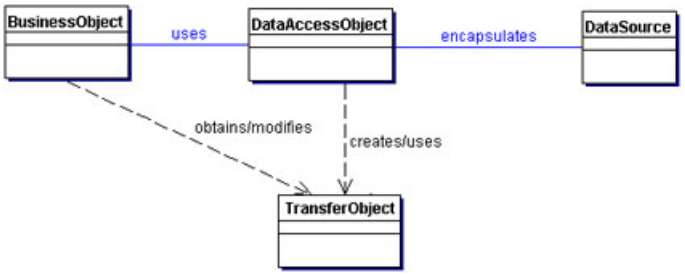


FIGURA 13

BusinessObject: representa el cliente de datos. Es el objeto que requiere el acceso a la fuente de datos para obtener y almacenar la información.

DataAccessObject: abstrae la implementación del acceso de datos para permitir un acceso transparente. El BusinessObject delega las operaciones de carga y actualización al DAO.

DataSource: representa la implementación de la fuente de datos. Puede ser un sistema relacional, una base de datos orientada a objetos, un repositorio XML o, incluso un sistema externo

TransferObject: representa el contenedor de información. El DAO puede usar TransferObjects para retornar información al cliente. También puede recibir datos del cliente encapsulados en un TransferObject con el fin de actualizar la información a la fuente de datos.

El siguiente diagrama de secuencia, muestra como interactúan estas clases con el fin de recuperar los datos.

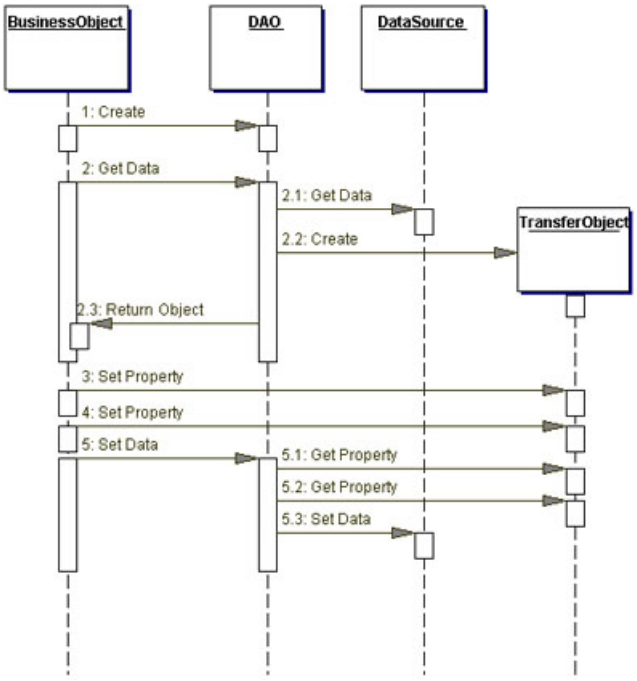


FIGURA 14

En primer lugar el BusinessObject invoca el método correspondiente del DAO. A continuación, este accede al DataSource y crea los TranseferObject que almacenaran la información con el fin de enviarla al BusinessObject.

Por tanto el procedimiento a seguir será crear una clase DAO para cada entidad persistente del modelo, de manera que esta gestione el acceso, ya sea para consultar o modificar datos. Algunas entidades tienen consultas con varias condiciones para optimizar las operaciones en la capa de negocio pero todas tienen unos métodos genéricos que pueden ser incluidas en una interfaz genérica. Los métodos son los siguientes:

create: crea una entidad persistente

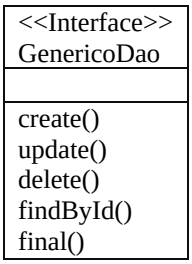
update: realiza la actualización de los atributos de la entidad

delete: elimina la entidad

findById: busca la entidad que tenga como clave primaria la pasada por parámetro.

findAll: busca todas las entidades de una misma clase

De esta forma separamos completamente las clases de negocio de la persistencia. La siguiente figura muestra la interfaz genérica:



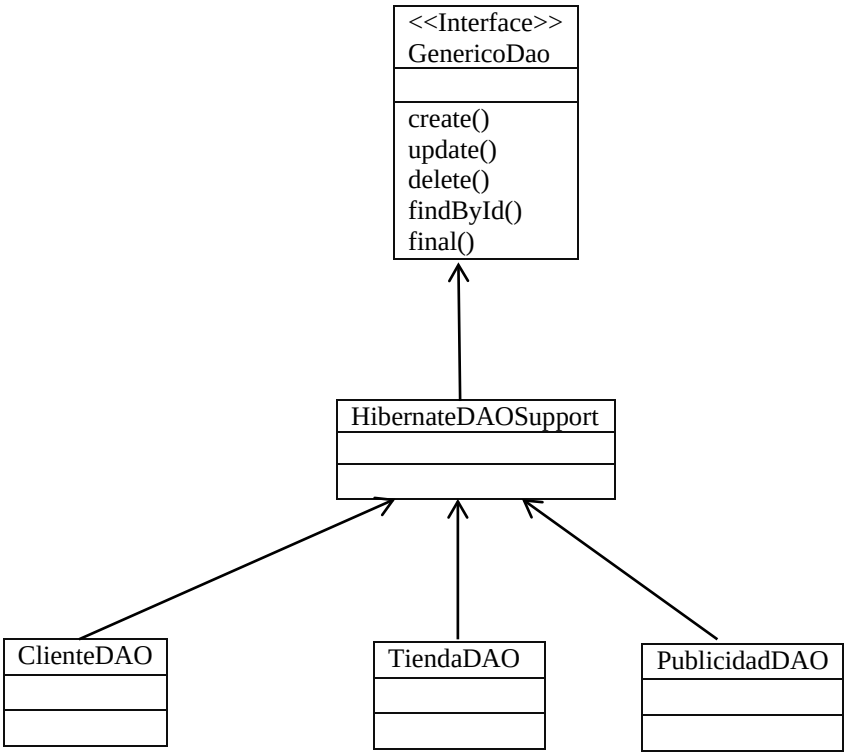
5.3.3.2 Mapeo Objeto-Relacional (ORM)

La representación física en un sistema relacional es completamente diferente a la red de objetos que usamos en aplicaciones orientadas a objetos. Esta diferencia se conoce como "desajuste en el paradigma objeto/relación". Tradicionalmente este desajuste había estado subestimado tanto en coste como en importancia y las herramientas para solucionarlo eran insuficientes. ORM es el nombre que se dio a las soluciones de mapeo automatizados al problema del desajuste objeto/relación. Las aplicaciones intermedias, middlewares, ORM se pueden considerar menos costosas y dependientes de productos concretos, más funcionales y más capacidades para superar cambios en los objetos internos o del esquema lógico subyacente.

Hibernate es un proyecto Open Source que representa una completa solución al problema de la gestión de datos persistentes en Java. Hace de intermediario entre la aplicación y la base de datos relacional y, además, permite al desarrollador centrarse en la lógica de negocio.

Hibernate aporta su propia API para trabajar, y por tanto, necesitamos algún mecanismo que permita integrar fácilmente nuestra capa de persistencia basada en DAO. La solución para combinar Hibernate con los DAO será utilizar el patrón Adaptador para construir una clase genérica que haga de puente entre la interfaz GenericDAO y la propia API d'Hibernate. La clase resultante es HibernateDaoSupport. Para cada concepto, crearemos la clase DAO que extienda de HibernateDaoSupport para poder utilizar todos los métodos implementados en ésta y dejar únicamente que los DAO proporcionen métodos para a ciertas funcionalidades específicas.

La siguiente figura nos muestra el diagrama completo de la capa de persistencia.



5.3.3.3 Modelo de datos

La base de datos esta diseñada de la siguiente manera.

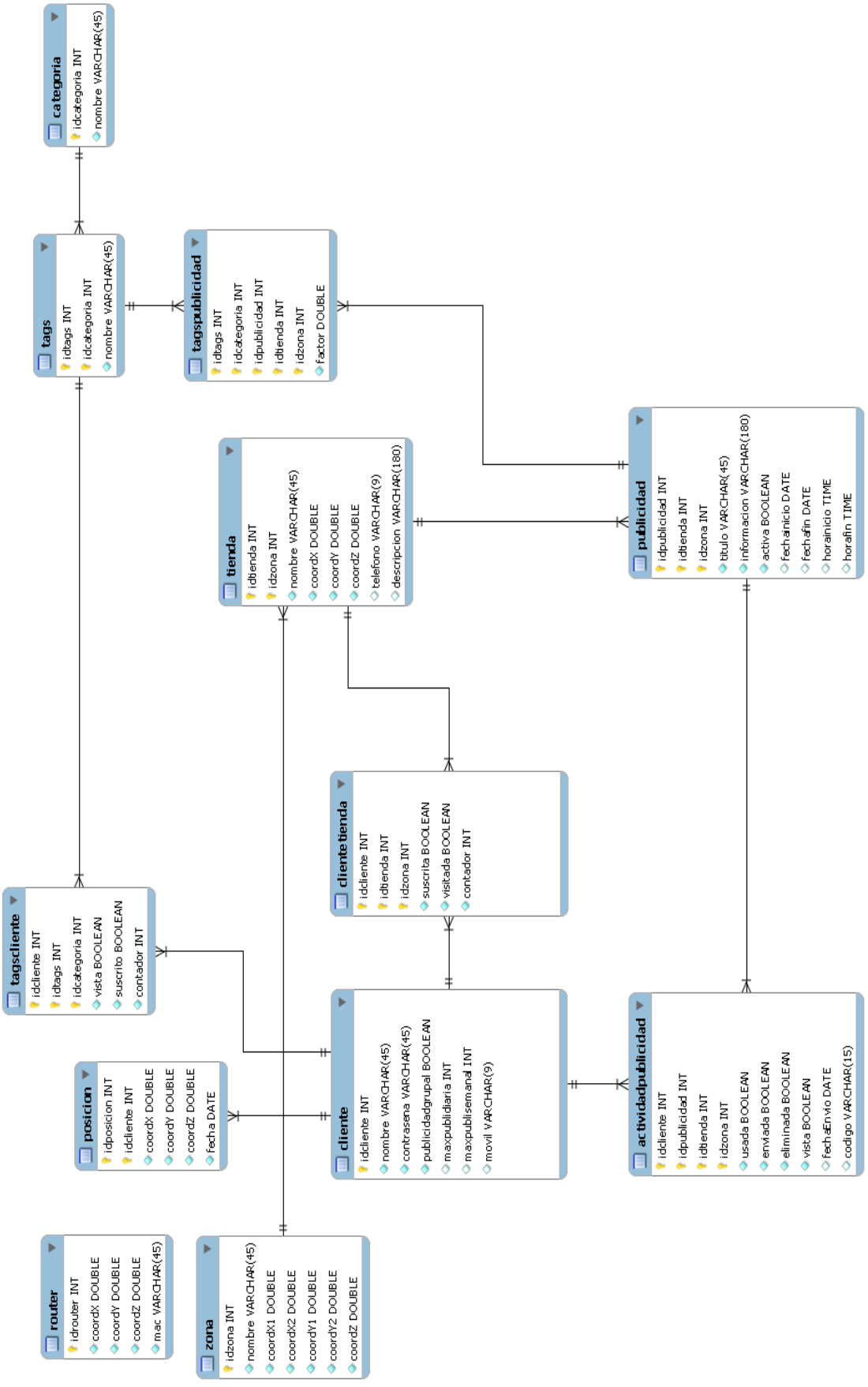

categoria
idcategoria INT
nombre VARCHAR(45)
tags
idtags INT
idcategoria INT
nombre VARCHAR(45)
tagspublicidad
idtags INT
idcategoria INT
idpublicidad INT
idtienda INT
idzona INT
factor DOUBLE
tienda
idtienda INT
idzona INT
nombre VARCHAR(45)
coordX DOUBLE
coordY DOUBLE
coordZ DOUBLE
telefono VARCHAR(9)
descripcion VARCHAR(180)
publicidad
idpublicidad INT
idtienda INT
idzona INT
titulo VARCHAR(45)
informacion VARCHAR(180)
activa BOOLEAN
fechainicio DATE
fechafin DATE
horainicio TIME
horafin TIME
clientetienda
idcliente INT
idtienda INT
idzona INT
suscrita BOOLEAN
visitada BOOLEAN
contador INT
tagscliente
idcliente INT
idtags INT
idcategoria INT
vista BOOLEAN
suscribo BOOLEAN
contador INT
posicion
idposicion INT
idcliente INT
coordX DOUBLE
coordY DOUBLE
coordZ DOUBLE
fecha DATE
cliente
idcliente INT
nombre VARCHAR(45)
contrasena VARCHAR(45)
publicidadgrupal BOOLEAN
maxpublidiaria INT
maxpublisemanal INT
movil VARCHAR(9)
actividadpublicidad
idcliente INT
idpublicidad INT
idtienda INT
idzona INT
usada BOOLEAN
enviada BOOLEAN
eliminada BOOLEAN
vista BOOLEAN
fechaEnvio DATE
codigo VARCHAR(15)
router
idrouter INT
coordX DOUBLE
coordY DOUBLE
coordZ DOUBLE
mac VARCHAR(45)
zona
idzona INT
nombre VARCHAR(45)
coordX1 DOUBLE
coordX2 DOUBLE
coordY1 DOUBLE
coordY2 DOUBLE
coordZ DOUBLE

FIGURA 15

5.4 Algoritmos

Un algoritmo se define como un conjunto de reglas establecidas en la programación de un sistema de gestión orientadas a la consecución de los objetivos previamente definidos. En la comercialización de los medios tienden a la consecución del máximo ingreso posible para el medio y en otros entornos buscan maximizar la diferencia entre los cotes de la compra y los ingresos obtenidos por la venta (IAB, 2012).

En este proyecto, para hacer que el envío de la publicidad se realice de forma inteligente se crean dos algoritmos, el algoritmo de perfil que crea un conocimiento de los clientes y el algoritmo de publicidad que toma las decisiones de que publicidad debe recibir cada cliente basándose el conocimiento de ellos y otras series de características.

5.4.1 Algoritmo de perfil.

Cuando el cliente móvil se ha registrado no se dispone de un perfil almacenado en el sistema y se carga un perfil inicial, que no dispone de algún conocimiento del usuario en particular. En este caso, el algoritmo asume que le puede gustar todo y esperará a recibir datos sobre su actividad para ajustar su conocimiento de él. Una vez se reciban datos de perfil este conocimiento se redefinirá de la siguiente manera.

Para crear el perfil *(caso de uso: algoritmo de perfil)*, se tiene en cuenta tres casos:

1. Los cambios de preferencias fijadas por el cliente móvil.
2. Las tiendas visitadas dentro del centro comercial, frecuencia con la que esta en cada zona de este y la visión por pantalla de la publicidad recibida.
3. El uso real de las promociones en las tiendas.

De manera opcional, los clientes tienen la capacidad para indicar aquello que le gusta. Si el cliente móvil define alguna etiqueta que refleje sus intereses, esta tendrá una gran relevancia pues refleja el tipo de publicidad que este desea recibir (*Caso de uso: actualizar perfil*). Lo mismo sucede cuando se suscribe a una tienda en especial, interesa más recibir publicidad de esta tienda sobre otras que ofrecen productos parecidos (*Caso de uso: suscribirse)*.

Cuando un cliente móvil entra en una tienda es porque muestra cierto interés por los productos que ofrecen. Este interés debe ser cuantificado en un orden de relevancia medio alto ya que si bien no garantiza que vaya a comprar algún producto cuando visita una tienda, es muy posible que se muestre receptivo a recibir publicidad sobre esta tienda que le terminen

por convencer. Para evaluar las tiendas visitadas, se utiliza las posiciones de coordenadas almacenadas del cliente móvil *(caso de uso: posicionamiento)* y comprueban si se encuentran dentro del rango de las coordenadas de alguna tienda del centro comercial y en caso afirmativo se registra la visita. Para identificar las tiendas visitadas que realmente le interesan al cliente se utiliza un contador que mide el número de veces que ha estado en la misma. Esto también permite eliminar márgenes de error que puedan surgir por visitas de clientes esporádicas y que muy probablemente no vuelva porque sus productos dejaron de interesarle.

Cuando el cliente móvil recibe una notificación de una nueva publicidad en el dispositivo este tiene la opción de aceptar o rechazar su visualización y solicitar el código de la promoción *(Caso de uso: ver código)*, entendiéndose que si sucede esta situación es porque el cliente móvil tiene cierto grado de interés o curiosidad por su contenido. Por este motivo el algoritmo estima que tanto la tienda que emite la publicidad como el contenido de esta son relevantes para el perfil.

Por último, la parte más importante del algoritmo que ayuda a generar conocimiento del usuario es cuando el cliente móvil usa el código promocional de una *publicidad (caso de uso: validar código)*. Esto significa un éxito de esta y por tanto ese tipo de publicidad encaja perfectamente en su perfil. El algoritmo se encarga de identificar los atributos de esta publicidad y asociarlas al perfil ya que existe una alta probabilidad de que este interesado en publicidades similares.

5.4.1.1 Diseño del algoritmo

Sean C, Ta, P, Ti y Z los nodos de las categorías, etiquetas, publicidades, tiendas y zonas que tiene el perfil asociado respectivamente, se asignan relevancias ‘r’ de acuerdo a la interacciones del cliente especificadas en el algoritmo (atributo de peso en minería de datos), siendo todo aquello que el cliente indique explícitamente lo mas relevante y luego lo que el propio algoritmo deduzca. Luego se realiza un enlace de nodos basada en la relación que existe entre ellos de acuerdo a la información que se dispone de las promociones, etiquetas y tiendas formando una red.

Se introduce ahora la formula usada para calcular la relevancia junto con el orden que existe entre ellos para identificar los mas relevantes. Esta se basa en los siguientes conceptos básicos:

- Las etiquetas en un perfil se puede clasificar en tres tipos: las que son *suscritas* (Tgs) directamente por el cliente, las *asociadas* (Tga) a publicidades vistas y las que no

pertenecen a ninguna de las dos anteriores (pertenecen a publicidades eliminadas por el cliente, es una etiqueta de alguna tienda que ha visitado, etcétera), las *irrelevantes*[2] (Tgi). El orden de relevancia es por este orden: Tgs > Tga > Tgi.

- Las tiendas de un perfil se clasifican en tres tipos: las que son *suscritas* (Ts) directamente por el cliente, las *visitadas* (Tv) físicamente y las *irrelevantes* (Ti), de la cual pudo recibir alguna publicidad pero la ignora. El orden de relevancia es por este orden: Ts > Tv > Ti.

- Las publicidades de un perfil se clasifican en tres tipos: las *usadas* (Pu) en la tienda y por tanto tiene un alto grado de efectividad, las publicidades *vistas* (Pv) por el cliente y las *irrelevantes* (Pi), que son las eliminadas o no vistas. El orden de relevancia es por este orden: Pu > Pv > Pi.

- Las publicidades tienen un conjunto de etiquetas asociadas que reflejan el contenido de la misma y es enviada por alguna tienda del centro comercial.

Dado que la relevancia de una etiqueta, tienda o publicidad depende del origen o interacción con esta, los valores iniciales (factores de relevancia) que reflejaran su importancia serán: $\alpha > \beta > \gamma$, siendo los pesos de estos 0.6, 0.3 y 0.1 respectivamente. Por lo tanto, Tgs, Ts y Pu tienen un factor de relevancia α, Tga, Tv y Pv tienen un factor de relevancia β y Tgi, Ti y Pi tienen un factor de relevancia γ. Estos pesos están dentro de una escala de 1 y se explica en el numero de veces se debe repetir un hecho para tener la misma relevancia que el que el máximo factor de relevancia. De esta forma, por ejemplo en las tiendas, tendrán la misma relevancia una tienda suscrita que el cliente no haya visitado (α) y una tienda que fue visitada dos veces (β), ya que $\alpha = 2\ \beta$. El motivo por el cual las irrelevantes tienen peso 0.1 se explica en que asignando factores de relevancia bajos a las publicidades que no gustan se puede obtener conocimiento no solo de aquello que gusta al cliente sino también de lo que no le gusta como se detalla en la formula de más abajo (un factor de relevancia 0 anularía las multiplicaciones de factores de relevancia).

La obtención de la relevancia se divide en dos fases. En la primera se calculan las relevancias iniciales de las etiquetas, tiendas y publicidades asociadas al perfil según el tipo que sea según la explicación de conceptos básicos anterior, así todos los elementos tienen un valor definido.

En la segunda fase se utiliza el hecho que se sabe qué etiquetas tiene una publicidad y qué tienda lo envió para recalcular la relevancia de las etiquetas y tiendas respecto al uso de la publicidad asociada. La relevancia global de una etiqueta o tienda viene dada por el contador

[2] Por *irrelevantes* se entiende que desde la perspectiva del cliente le resulta indiferente o bien tiene un desconocimiento de ella, pero para el algoritmo sí es importante para saber con mayor certeza lo que podría interesarle y lo que no.

que indica cuantas veces apareció en una publicidad. Luego, para calcular su relevancia real respecto se procede de la siguiente forma:

Dado que las relevancia globales es equivalente a al contador de apariciones se necesita un método que permita evaluarlas dentro de un rango fijo, por ejemplo [0..1]. Por este motivo, se agrupa todas etiquetas y tiendas por el tipo al que pertenezcan en un vector y se normalizan dividiéndolas por el valor máximo del grupo al que pertenezcan. De esta forma el sumatorio de todo el grupo es igual a 1.

$$Sea\ Tg_s(x_{1,}..,x_n)\ las\ relevancias\ de\ las\ etiquetas\ suscritas\ t_1..t_n\ y\ x_i\ la\ de\ máximo\ valor, las\ relevancias\ normalizadas\ son\ x'_j = \frac{x_j}{x_i}\ para\ j \in 1..n\ y\ 1 = \sum Tg'_s = \sum_1^n x'_i$$

A estos vectores con las relevancias normalizadas se le aplica el factor de relevancia α, β, γ según su tipo para que estas reflejen la importancia que tienen para el cliente.

$$Sea\ Tg'_s, Tg'_a, Tg'_i\ las\ relevancias\ de\ las\ etiquetas\ normalizadas\ por\ tipo\ y\ sus\ factores\ de\ relevancia\ \alpha,\beta,\gamma\ respectivamente:$$

$$Tg''_s = \alpha * Tg'_s, Tg''_a = \beta * Tg'_a, Tg''_i = \beta * Tg'_i\ y$$

$$1 = \alpha * \sum Tg'_s + \beta * \sum Tg'_a + \gamma * \sum Tg'_i$$

$$El\ mismo\ procedimiento\ se\ aplica\ para\ las\ tiendas\ T'_s, T'_v, T'_i.$$

Como estas etiquetas y tiendas están asociadas a una publicidad enviada al cliente móvil se tiene que tener en cuenta que uso final tuvo para este (si la llego a usar, solo la vio o la descartó). Dependiendo de esto la relevancia final puede variar considerablemente (por ejemplo, un cliente esta suscrito a dos tiendas pero solo usa la publicidad de una de ellas, por lo que claramente una tienda es más atractiva que la otra). Por ello se debe aplicar el factor de relevancia correspondiente del tipo de publicidad las etiquetas y tiendas que la incluyen.

$$Se\ multiplica\ a\ \left(Tg''_s, Tg''_a, Tg''_i\right)\ y\ (T''_s, T''_a, T''_i)\ por\ el\ factor\ de\ relevancia\ \alpha,\beta,\gamma\ según\ pertenezcan\ a\ una\ publicidad\ usada, vista\ o\ eliminada\ respectivamente\ estos\ dos\ vectores\ pueden\ pertenecer\ a\ más\ de\ un\ tipo, por\ ejemplo\ pueden\ ser\ vistas\ y\ elminadas\ o\ vistas\ y\ usadas. De\ esta\ forma\ el\ sumatorio\ puede\ estar\ entre\ el\ rango\ 0\ a\ 1. La\ formula\ para\ establecer\ que\ los\ sumatorios\ solo\ pueden\ estar\ entre\ el\ rango\ especificado\ es\ la\ siguiente:$$

$$Relevancia\ (etiquetaSuscrita|cliente)$$
$$= \sum publicidadUsada\ + \sum publicidadVista$$
$$+ \sum publicidadIrrelevante$$
$$= \left(\left(\frac{n_u}{n_m}\right) * \alpha * \sum Tg''_s\right) + \left(\left(\frac{n_v}{n_m}\right) * \beta * \sum Tg''_s\right)$$
$$+ \left(\left(\frac{n_i}{n_m}\right) * \gamma * \sum Tg''_s\right)$$

$donde\ n_u, n_v, n_i\ es\ el\ numero\ de\ veces\ que\ aparce\ la\ etiqueta\ en\ una\ publicidad\ usada, vista\ o\ irrelevante\ respectivamente\ y\ n_m = \max(n_u, n_v, n_i)$.

Continuamente el cliente puede cambiar de gustos e intereses y analizar el uso que hace de la publicidad que se le enviar cambia las relevancias de sus preferencias ajustándose constantemente a los que mas le puede resultar atractivo en el momento.

Estas relevancias calculadas cambian constantemente y en consecuencia sus intereses, pero se puede dar el caso que este cambio de intereses se produzca durante una franja fija de tiempo dentro del año (por ejemplo, en temporada de reyes se compran productos que en otros días del año no se comprarían) por lo que para intentar localizar estas situaciones se almacena un registro de las relevancias resultantes del perfil cada semana y la fecha que se calculo. La relevancia final de un elemento del perfil es la media de la relevancia actual y la relevancia para ese mismo elemento en años anteriores con pesos 2 y 1 respectivamente para que las relevancias guardadas no afecten mucho al último calculo hecho.

rx=(2 rx + registro_relevancia(ix))/3 para x ϵ 1,2,3….max_elementos

La relevancia de las categorías y las zonas dentro de un perfil es el sumatorio de las etiquetas y tiendas que la componen respectivamente.

$r(Cx) = \frac{\sum_1^n r(Tg_i)}{n}\ donde\ n\ es\ el\ numero\ de\ etiquetas\ en\ el\ sistema.$

$r(Zx) = \frac{\sum_1^m r(T_i)}{m}\ donde\ m\ es\ el\ numero\ de\ tiendas\ en\ el\ sistema.$

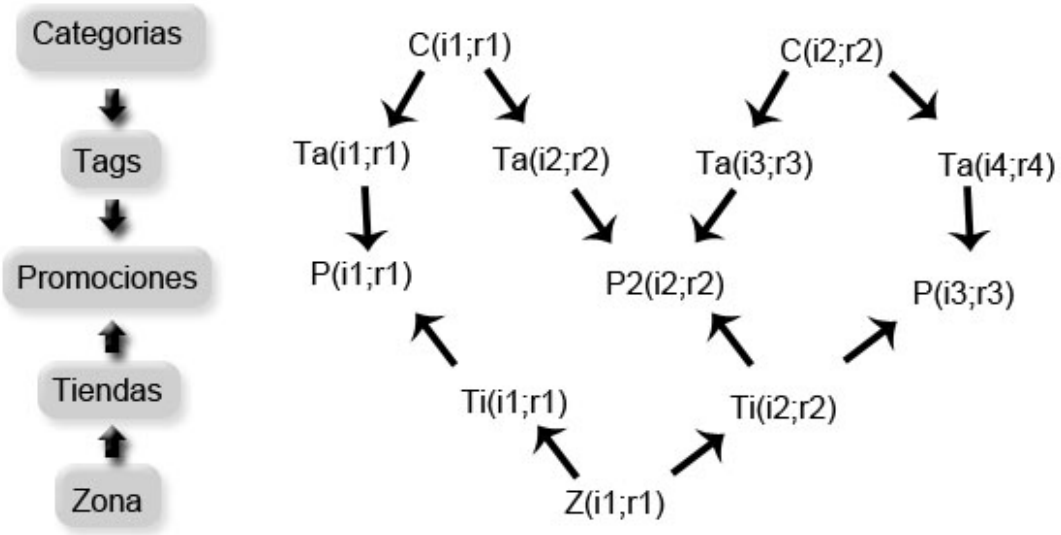


FIGURA 16

Por ultimo se utiliza el método, en minería de datos, de aprendizaje por reglas de asociación en las etiquetas y tiendas para identificar aquellas que aun no existen en perfil pero que tienen una alta probabilidad de interesarle en base a la información que se tiene de otros perfiles en sistema.

Para este caso, las relevancias se dividen en 5 categorías: nada, poca, normal, mucha y bastante. Cada relevancia se sitúa en la sub-categoría que le corresponde dependiendo de su valor, de la siguiente forma:

Nada = 0 … max/5
*Poca = max/5 … 2*max/5*
*Regular= 2*max/5 … 3*max/5*
*Bastante= 3*max/5 … 4*max/5*
Mucho = 4 max/5 … max* Siendo *max* el máximo valor de relevancia en el perfil.

Cada ítem de este algoritmo corresponde a un par {categoría_relevancia, etiqueta} y {categoría_relevancia, tienda} para la ejecución en etiquetas y tiendas respectivamente. Un ejemplo de esto se puede ver en la siguiente tabla:

Id Cliente	Ta(i1), poca	Ta(i1), normal	Ta(i2), poca	Ta(i2), normal
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0

Algoritmo_perfil (Perfil)

Si perfil = null;
Perfil = perfil_inicial() ;
Fsi

*/*Carga las etiquetas, tiendas y publicidad que con la que cliente interactuó de alguna manera, la relevancia inicial y la relevancia final (igual a relevancia inicial la primera vez que se calcula)-Algoritmo principal*/*
Tg <- Set { etiqueta, relevancia_inicial ,relevancia}
Ti <- Set { tienda , relevancia_inicial ,relevancia }
P <- Set { publicidad , relevancia_inicial }

Para cada tg ∈ Tg hacer
Si p->etiquetai existe entonces
*Ta->etiquetai->relevancia = Ta->etiquetai->relevancia + Ta-> etiquetai->relevancia_inicial * p->relevancia_inicial;*
Fsi
Si p->tiendai existe entonces
*Ti->tiendai->relevancia = Ti->tiendai->relevancia + Ti-> tiendai->relevancia_inicial * p->relevancia_inicial;*
Fsi
Fpara

/ Actualiza la relevancia las registradas en la misma semana de años anteriores*/*
Mientras existe Ta->etiquetai hacer
*Ta->etiquetai->relevancia= (2*Ta->etiquetai->relevancia + registro_relevancia(Ta->etiquetai))/3*
i++;
Fmientras

Mientras existe Ti->tiendai hacer
*Ti->tiendai->relevancia= (2*Ti->tiendai->relevancia + registro_relevancia(Ti->tiendai))/3*
i++;
Fmientras

/ Actualiza las relevancia de las categorías y zonas */*

Para cada ci ∈ C hacer
ci-> relevancia = $\sum_{j=1}^{n}$ *Ta->etiquetaj->relevancia;*
Fpara

Para cada zi ∈ Z hacer
zi-> relevancia = $\sum_{j=1}^{n}$ *Ti->tiendaj->relevancia;*
Fpara

Perfil->Ta= Ta;
Perfil->Ti= Ti;
Perfil->C = C;
Perfil->Z= Z;

Return Perfil;

5.4.2 Algoritmo de publicidad.

El algoritmo de publicidad *(caso de uso: algoritmo de publicidad)* se encarga de utilizar toda la información que dispone de los clientes móviles para decidir que publicidad enviarle a cada uno en un momento determinado. A este proceso de encontrar la publicidad que mejor se ajusta a cada cliente móvil de forma inteligente se le llama *matching*. El *matching* se realiza en dos formas, en activo y en pasivo. Al poder existir más de una publicidad que puede encajar con un cliente móvil en un momento determinado se utiliza un orden de *prioridad* para establecer cual debería enviarse primero en caso de no poder enviarle todas.

El algoritmo realiza un *matching en pasivo* cuando se utiliza toda la información suministrada por el *gestor de conocimiento* y el *gestor de mapa* para ejecutar un algoritmo de minería de datos de clasificación probabilística de los clientes móviles dentro de las publicidades disponibles. Es decir, clasifica a los clientes móviles según la publicidad que mejor encaja con ellos.

Este proceso necesita un tiempo relativamente largo de ejecución por lo que puede estar programado para realizarse en una hora exacta del día (por ello no toma en cuenta la actividad actual del cliente móvil) y guardar este conocimiento para la próxima vez que el cliente móvil entre en el centro comercial o sobrescribirlo si se recibió nueva información del *gestor de conocimiento*.

El algoritmo realiza un *matching en activo* cuando el cliente móvil se encuentra dentro del centro comercial y se recibe continuamente información sobre su actividad actual. El algoritmo dispone del conocimiento del *matching pasivo* pero reajusta este conocimiento con los datos que va recibiendo en tiempo semi-real. Los casos que permiten este reajuste son los siguientes:

1. El centro comercial esta dividido por zonas especificas (pe: zona alimentación, zona moda, zona deportiva, etcétera), en cuanto recibe notificación de que un cliente móvil ha cambiado de zona, aumenta la prioridad de las publicidades de las tiendas que pertenecen a esa zona.

2. Cuando recibe una notificación de que el cliente móvil se encuentra dentro de una tienda durante un tiempo relevante, selecciona las publicidades de estas tiendas con alta prioridad.

3. Activa las publicidades que solo son efectivas durante la franja horaria actual (pe: restaurantes y la hora de comer) aumentando su prioridad y descarta aquellas que no deben enviarse en ese horario.

El orden de prioridades en el supuesto de que se cumplan varios casos es: 2 > 3 > 1. Una vez asignadas las publicidades prioritarias se pueden dar dos situaciones: Actualmente hay mucha gente en el centro comercial o más bien hay poca. Si se da el primer caso, el algoritmo selecciona para un grupo de usuarios, aquellas publicidades que coincidan para todos intentando que estén entre las más prioritarias de la mayoría. Si se da el segundo caso, el algoritmo elige la publicidad con más prioridad para cada cliente móvil.

El algoritmo lo realiza así porque el sistema necesitaría un tiempo considerable para enviar publicidad en un momento determinado a una gran cantidad de clientes móviles y hasta que se complete el envío puede que este ya se encuentre en otro sitio y no le interese volver.

Por ultimo, como una medida para controlar el Spam, el algoritmo tiene un limite de envió de publicidad por cliente móvil que no puede exceder de cierta cantidad al día o semana. Este límite es fijado por el cliente móvil o se usa uno predefinido en caso contrario. Además puede usar las opciones de configuración para indicar si prefiere recibir la publicidad en grupo (cuyo límite lo define él mismo) o uno a uno.

5.4.2.1 Diseño del algoritmo

Se plantea la implementación del algoritmo para realizar el matching en pasivo de la siguiente forma. Dado que las publicidades tienen asociadas una tienda que la crea y un conjunto de etiquetas que la definen, la relevancia real de una etiqueta puede verse afectada por la importancia que tiene la tienda en el conocimiento del cliente. Por este motivo, ya no se quiere la relevancia de una etiqueta en un cliente sino la relevancia que tiene una etiqueta para un cliente en una tienda concreta, que es la que envía la publicidad.

Sea P el conjunto de todas las publicidades que dispone el sistema en el momento actual, p_i es el subconjunto de publicidades para un cliente i.

Sea TG el vector relevancias de las etiquetas existentes en el sistema de longitud m y T el vector de relevancias de las tiendas en el centro comercial de longitud n. A es la matriz, de tamaño mxn, resultante de multiplicar TG*T.

$$A = \begin{pmatrix} tg_i * t_j & \cdots & tg_i * t_n \\ \vdots & \ddots & \vdots \\ tg_m * t_j & \cdots & tg_m * t_n \end{pmatrix} donde\ tg_i \in TG\ y\ t_j \in T$$

De esta forma, usando las relevancias de las etiquetas y tiendas proveídas por el algoritmo de perfil se puede construir una matriz donde el producto estas relevancias muestra el peso especifico que una etiqueta tiene en una tienda. Siendo A la matriz, A_i es el vector de etiquetas en la fila *i* de la matriz A que tiene como tienda a asociada t_i.

Dado que el rango de valores de TG y T son [0..1] según el algoritmo de perfil, la multiplicación de ambos seguirá estando dentro del rango [0..1]

$Sea\ P\ el\ conjunto\ de\ publicidades\ que\ tiene\ sistema\ en\ ese\ momento.\ A\ cada\ publicidad\ p_i \in P\ se\ le\ asigna\ una\ vector\ V = (x_1, x_2, .., x_n)\ de\ longitud\ m\ donde\ cada\ valor$

$x_i\ es\ de\ la\ forma$:

$$x_i = \begin{cases} 1\ si\ tiene\ asociada\ una\ etiqueta\ tg_i \\ 0\ en\ cualquier\ otro\ caso \end{cases}$$

El método usado para saber que publicidad se debe enviar a un perfil usando el *matching pasivo* es usando la similaridad de cosenos, ya que no solo toma en cuenta los pesos de las etiquetas que coincidentes entre los perfiles y las publicidades sino que también toma en cuenta las etiquetas no coincidentes para el calculo. Este método es usado en minería de datos para calcular la cohesión, en nuestro caso entre publicidades y perfiles.

Algoritmo_publicidad()

TG <- {etiquetas,relevancias}
T<- {tiendas, relevancias}
*A=TG*T;*
Para cada per ∈ Perfil hacer
    Para cada p_i ∈ P hacer
    j <- j-fila de la tienda $p_i \rightarrow$ tienda;
    $d = (A_j * V)/|A_j| * |V|$;
    / μ *es el mínimo valor (peso) entre [0..1] necesario para incluir la publicidad en el perfil, por ejemplo 0.6*/*
    Si $d >$ μ entonces incluir p_i en per
    Fpara
Fpara

Una vez cada perfil tiene la publicidad asociada se utiliza el matching activo para evaluar cual tiene mas prioridad sobre otras dependiendo del caso en que se encuentre.

En el caso que el cliente cambie de zona se utiliza el vector de relevancia de Zonas que el algoritmo de perfil generó. El envío de publicidad solo procederá si la relevancia de la zona para ese perfil es mayor o igual a 0.5. De esta forma se evita colapsar de publicidad al cliente que pasa a través de una zona que ofrece productos que es muy posible que no le interesen. Esto también hace una distinción entre cambios a una zona que hace el cliente para comprar en ella y cuando cambia de zona como simple intermediaria o paso para llegar a la zona que realmente le interesa.

Z<- {zonas, relevancias}

Normalizar(Z);

i<- identificador nueva zona; Si Z_i>0.5 *entonces Para* cada p∈perfil→publicidad hacer *Si* $p \rightarrow tienda$ esta en Z_i *entonces* $p \rightarrow prioridad$=3;Fsi

Fpara

Fsi

En el caso de que se encuentre dentro de una tienda durante un tiempo relevante (por ejemplo: 1 minuto) el proceso es parecido pero en este caso que en el anterior pero en este caso se asigna una prioridad mayor a las publicidades enviadas por la tienda en la que se encuentra actualmente. La prioridad es mayor para dar más importancia a lo que espacialmente esta más cerca del cliente (una tienda) sobre un conjunto más grande (la zona sobre la que se encuentra).

T<- {tiendas, relevancias}

Normalizar(T);

i<- identificador tienda; Si T_i>0.5 *entonces Para* cada p∈perfil→publicidad hacer *Si* $p \rightarrow tienda = T_i$ *entonces* $p \rightarrow prioridad$=4;Fsi

Fpara

Fsi

En el caso de aquellas publicidades que solo se activan durante una franja horaria, se utiliza el reloj del sistema para notificar al algoritmo de este hecho. Por lo tanto, el sistema tiene registrado un los horarios de activación y desactivación de las publicidades activas y cada vez que se llega a la hora determinada se envía una alerta para activar o desactivar la publicidad según corresponda. A su vez ejecuta el algoritmo_publicidad() para hacer el matching de esta única publicidad con los clientes.

5.5 Elección de la tecnología

5.5.1 El SO móvil

En el desarrollo de este proyecto se ha pensado que la aplicación debe esta disponible al mayor número de personas posibles, por lo que se debe elegir un sistema operativo móvil con la mayor cuota de mercado posible en los smartphones. Por este motivo tanto Android como iPhone son los mejores candidatos.

Actualmente existe un incremento espectacular de venta de dispositivos que llevan Android y existen previsiones de que en un futuro no muy lejano este sea el sistema operativo número uno de smartphones en el mundo, incluido España. Por otro lado, iPhone no se queda atrás, goza de gran popularidad en el mercado y con el lanzamiento de su próximo Smartphone luchara mano a mano con los Android.

Dado que se dispone de un smartphone que utiliza Android para realizar la implementación y pruebas y que sus ventajas como en el lenguaje evitan tiempo de aprendizaje, para este proyecto se ha elegido desarrollar la aplicación usando este sistema operativo. A continuación vemos las características de los dos sistemas operativos más usados reflejando las ventajas y desventajas de cada uno:

5.1.1.1 Android

El sistema operativo fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005, más tarde Google liberaría el código de Android bajo licencia Apache. A modo de curiosidad, el sistema operativo está compuesto por 12 millones de líneas de código, incluyendo 3 millones de líneas de XML, 2.8 millones de líneas de lenguaje C, 2.1 millones de líneas de Java y 1.75 millones de líneas de C++.

Desarrollar en Android tiene muchas ventajas, a continuación se enumeran las principales:

Java: es el lenguaje que se utiliza para programar, por lo que no se necesitaría un tiempo extra para programar.

Software libre: cualquier programador puede realizar modificaciones a las partes más internas de un programa, a diferencia de productos cerrados que distribuyen el código bloqueado. Gracias a la licencia Apache, en la que se basa el sistema operativo Android, cualquier programador puede compartir, utilizar código, realizar versiones a nuevos sistemas, etcétera.

Accesibilidad: la participación de terceros desarrolladores es más accesible en el sistema operativo de Google. Las APIs de Android permiten crear cualquier cosa, facilitando el trabajo y motivando a los desarrolladores.

Gran comunidad de desarrolladores: posiblemente gracias al punto anterior existe una enorme comunidad de fans (fandroids) escribiendo aplicaciones y/o frameworks para extender las funcionalidades del sistema operativo.

Pese a ser muy ampliamente usado por los desarrolladores, Android también tiene algunas desventajas:

Fragmentación: Android ha sido criticado muchas veces por la fragmentación que sufren sus terminales al no disponer de actualizaciones constantes por los distintos fabricantes.

5.1.1.2 iPhone OS

El iPhone OS es parte del sistema operativo iOS, un derivado de Mac OS X, que a su vez está basado en Darwin BSD. Darwin es el sistema que subyace en Mac OS X y es un derivado del sistema UNIX. Darwin proporciona prestaciones modernas como la memoria protegida, la multitarea, la gestión avanzada de memoria y el multiproceso simétrico.

Desarrollar en iPhone tiene muchas ventajas, a continuación se enumeran las principales:

Framework fácil de usar: el framework de iOS, Cocoa-Touch, es realmente fácil de aprender y de utiliza, además, personalmente creo que el desarrollador disfruta programando y se siente motivado en aprender nuevos aspectos y conocimientos para mejorar las aplicaciones.

Gran comunidad de desarrolladores: como en Android este es un punto muy a favor de iOS. Tener una gran comunidad de desarrolladores hace que el desarrollador novato pueda acceder a una gran cantidad de información y de ejemplos para aprender. También ofrece a los desarrolladores más experimentados un punto de reunión para intercambiar opiniones y motivar a que Apple siga ofreciendo mejoras e innovaciones para el desarrollo de aplicaciones.

iOS no presenta muchas desventajas respecto al tema tecnológico, quizá de cara a un estudiante las más importantes hacen referencia al aspecto económico como ahora veremos:

Objective-C: si en Android programamos en Java, en iOS lo hacemos con Objective-C. Este lenguaje no es tan conocido ni está tan extendido como Java.

Desarrollo siempre bajo Mac OS X: una de las grandes desventajas que presenta ser desarrollador de iOS es que se necesita un ordenador Mac puesto que no es posible encontrar el IDE con todo el framework de iOS para cualquier otra plataforma. Si ya disponemos de un Mac de antemano esto no es un problema, pero si disponemos de un PC es un poco doloroso gastar dinero en otra máquina solo para desarrollar una aplicación para iOS. Además, si se quiere subir la aplicación a la Apple Store es necesario hacerlo desde un Mac ya que se requiere del número de serie de la máquina.

Cuota de desarrollador: un poco relacionado con el punto anterior en cuanto referente a la economía, es necesario pagar una cuota de 99$ anuales para disfrutar de las herramientas para desarrollar y poder subir aplicaciones a la tienda.

5.1.2 Spring

Spring Framework es una plataforma de Java que proporciona la infraestructura de apoyo global para el desarrollo de aplicaciones Java.

Spring se encarga de la infraestructura para que pueda centrarse en su aplicación, permitiendo la construcción de aplicaciones de "objetos típicos JAVA" (POJOs) y aplicar los servicios de la empresa de forma no invasiva. Esta capacidad se aplica al modelo de programación total de Java SE y parcial de Java EE.

El framework consiste de varias características organizadas en cerca de 20 módulos. Estos módulos están agrupados en Core Container, Data Acces/Integration, Web, AOP (Aspect Oriented Programing), Instrumentation y Test, como se muestra en el siguiente diagrama:

La misión principal de Spring es la de simplificar el desarrollo de aplicaciones Java y, a diferencia de otros frameworks, se puede acoplar a la aplicación si necesidad de modificar el código para utilizar las funcionalidades y beneficios que ofrece.

Para usar Spring no es imprescindible implementar un interfaz propio de Spring o heredar de una clase propia, lo que significa que Spring estará ahí pero las clases serán Java puro y duro. Como mucho, las clases tendrán anotaciones propias de Spring, pero no dejan de ser anotaciones por lo que la clase será reutilizable en cualquier momento, sin cambios.

Otro de los motivos por el cual se decidió utilizar Spring es porque su modulo ORM (parte de la capa Data Access /Integration) provee un capa de integración las API's de mapeo objeto-relacional mas populares, entre la que esta Hibernate, una tecnología también usada en este proyecto por las grandes ventajas que ofrece. Usando el paquete ORM de Spring se puede

usar Hibernate en combinación con todas las otras funcionalidades que Spring ofrece, como la gestión de transacción declarativa.

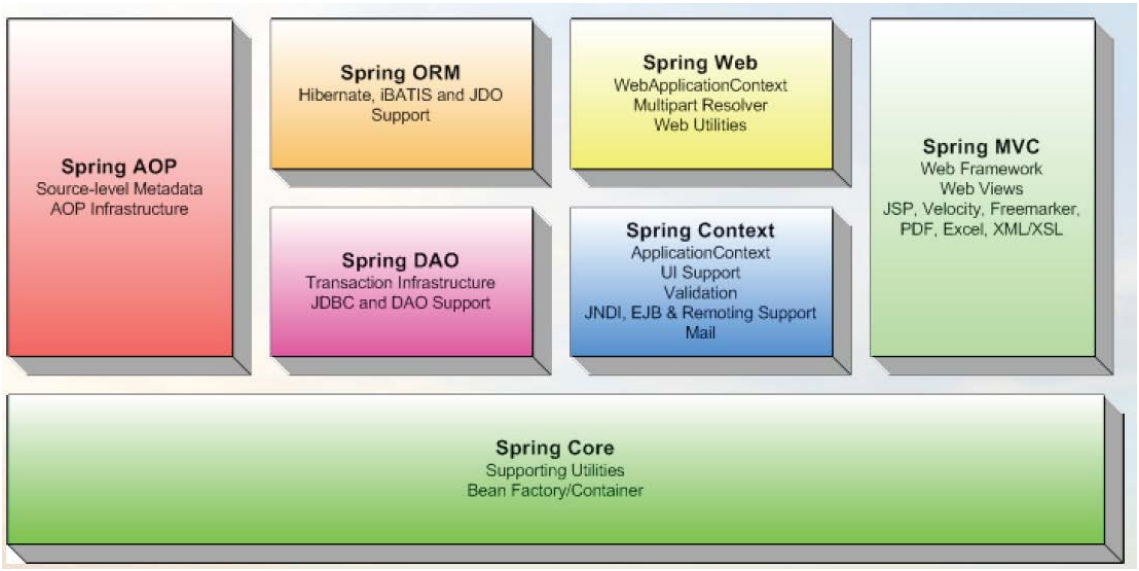


FIGURA 17

5.1.3 Hibernate

Hibernate es una herramienta de mapeo objeto-relacional (ORM) para la plataforma java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans[3] de las entidades que permiten establecer estas relaciones.

Usar Hibernate propone una serie de ventajas:

Modelo de programación natural: Hibernate permite desarrollar clases persistentes orientadas a objeto en java.

Persistencia transparente: Hibernate no requiere interfaces o clases base para las clases persistentes y permite que cualquier estructura de datos o clase sea persistente. Además, Hibernate permite construir los procedimientos más rápido.

Fiabilidad y escalabilidad: Hibernate es conocida por tener una excelente estabilidad y calidad. Fue diseñado para trabajar en un cluster de servidores de aplicaciones y ofrecer una arquitectura altamente escalable.

[3] También llamados JavaBeans, son un modelo de componentes creados para la construcción de aplicaciones en Java. Se usan para encapsular varios objetos en un único objeto, para hacer uso de un solo objeto en lugar de varios más simples.

Integración de consultas: Incluye compatibilidad con las consultas nativas de SQL.

5.1.4 MySQL

Es el sistema gestor de base de datos relacional más usada del mundo que se ejecuta en el servidor proveyendo acceso a muchos usuarios a un número de base de datos. El proyecto esta disponible bajo los términos de la licencia publica GNU y es a menudo usado por los proyectos open source de software libre como su sistema gestor de base de datos.

Es una base de datos popular para su uso en aplicaciones web y es usada en muchas de las páginas más frecuentemente visitadas. Tiene una fácil integración con Hibernate y además, como puntos fuertes, provee una gran velocidad en las transacciones mientras al mismo tiempo consume pocos recursos.

6 Implementación

La implementación es la etapa que sigue al diseño. En esta etapa se codifican todas las clases que se han diseñado hasta ahora y se han de cumplir todos los requerimientos establecidos. Es en este momento cuando se observa el comportamiento pensado para todo el sistema.

En este capitulo se muestran todas las características a nivel de implementación que forman parte del proyecto, el estudio de las tecnologías y las codificaciones que son mas relevantes.

6.1 Android

6.1.1 Arquitectura

Los componentes principales del sistema operativo Android:

- **Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

- **Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework).

- **Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

- **Runtime de Android**: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx".

- **Núcleo Linux**: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

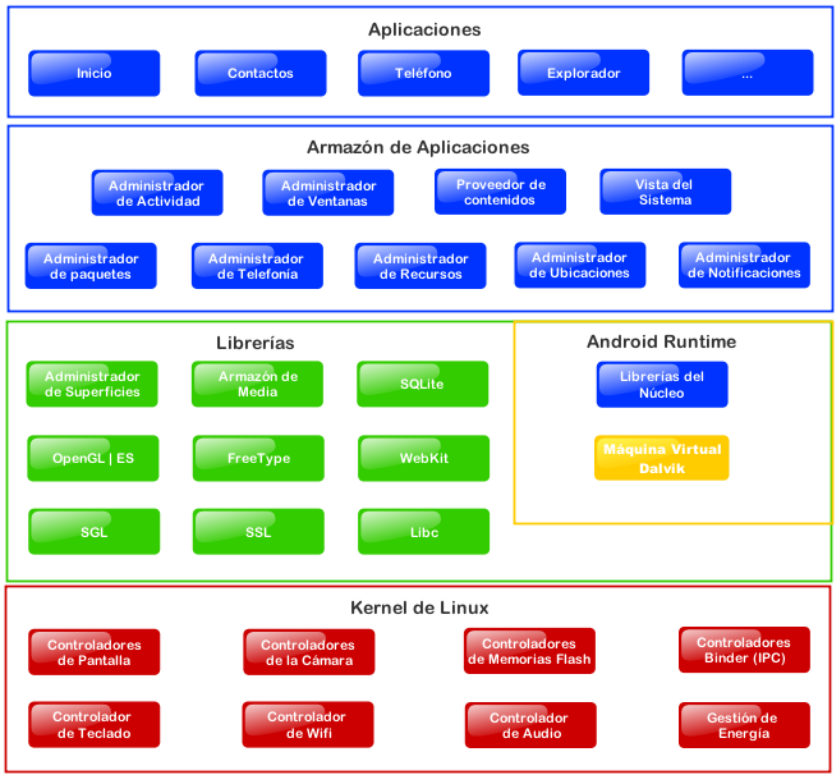


FIGURA 18

6.1.2 Componentes fundamentales

6.1.2.1 Activity

Las *Activity* (actividades) representan el componente principal de la interfaz grafica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una ventana en cualquier otro lenguaje visual.

6.1.2.2 View

Los objetos *View* son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, análoga por ejemplo a los controles de java o .NET. De inicio, Android pone a nuestra disposición una gran cantidad de controles básicos, como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear nuestros propios controles personalizados

6.1.2.3 Service

Los *servicios* son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema

operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales (activities) si se necesita en algún otro momento la interacción con el usuario

6.1.2.4 Content Provider

Un *content provider* es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de nuestra aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, nuestra aplicación podrá acceder a los datos de otra a través de los *content providers* que se hayan definido.

6.1.2.5 Widget

Los *Widgets* son elementos visuales, normalmente interactivos, que pueden mostrarse en la pantalla principal (home screen) del dispositivo Android y recibir actualizaciones periódicas. Permiten mostrar información de la aplicación al usuario directamente sobre la pantalla principal.

6.1.2.6 Intent

Un *Intent* es el elemento básico de comunicación entre los distintos componente Android que hemos descrito anteriormente. Se pueden entender como los mensajes o petición que son enviados entre distintos componente de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar un activity desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etcétera.

6.1.3 Mapa

Un aspecto importante de la aplicación móvil es que el cliente pueda disponer de un mapa donde encontrar las tiendas y que además pueda ubicarse en el mismo para saber como llegar. Para ello se ha creado un mapa prototipo donde se refleje la posición de las tiendas a nivel global y marcando con una “x” la posición actual del cliente móvil.

Android incluye un soporte para realizar gráficos 2D o 3D de alto rendimiento con la librería Open Graphics Library (OpenGL), específicamente OpenGL ES API. OpenGL es una API de gráficos multiplataforma que especifica una interface de software para el procesamiento de gráficos en 3D. OpenGL ES API es como la especificación OpenGL pero dirigido a hacia dispositivos embebidos.

Android soporta OpenGL a través de su framework API y del Native Development Kit (NDK). Hay dos clases funcionales en el framework de Android que nos permiten crear y manipular gráficos con OpenGL ES API: GLSurfaceView y GLSurfaceView.Renderer.

- **GLSurfaceView**: Esta clase es una View donde se puede dibujar y manipular los objetos usando llamadas a OpenGL y es similar en función a SurfaceView. Se puede usar esta clase creando una instancia de GLSurfaceView y añadiéndole un Renderer a ella.

- **GLSurfaceView.Renderer**: Esta interface define los métodos necesarios para dibujar gráficos en una GLSurfaceView. Se debe proveer una implementación de esta interface como una clase independiente y añadirla a una instancia GLSurfaceView usando GLSurfaceView.setRenderer().

La interface GLSurfaceView.Renderer requiere que se implementen los siguientes métodos:

- **OnSurfaceCreated():** El sistema llama a este método una vez, cuando crea la GLSurfaceView. Este método se usa para realizar las acciones que se necesitan que pasen una sola vez, como configurar los parámetros de entorno de OpenGL o inicializando los objetos gráficos de OpenGL.

- **OnDrawFrame():** El sistema llama a este método en cada redibujo de la GLSurfaceView. Este método se usa como el punto primario de ejecución para dibujar (y redibujar) los objetos gráficos.

- **OnSurfaceChanged():** el sistema llama a este método cuando la geometría de la GLSurfaceView cambia, incluyendo cambios en el tamaño de la GLSurfaceView o la orientación de la pantalla del dispositivo. Por ejemplo, el sistema llama a este método cuando el dispositivo cambia de posición horizontal a vertical. Este método se usa para responder a los cambios en el contenedor de la GLSurfaceView.

Haciendo uso de OpenGL para dibujar el mapa, creamos una GLSurfaceView y una implementación de GLSurfaceView.Renderer que añadiremos a nuestra GLSurfaceView. Este dibujo será lo que se muestre por nuestro dispositivo:

```
mGLView = new GLSurfaceView(this);
mRenderer = new Renderer(this);
mGLView.setRenderer(mRenderer);
mGLView.setKeepScreenOn(true);
setContentView(mGLView);
```

La implementación de la interface GLSurfaceView.Renderer se define en nuestra clase Renderer con todos los métodos necesarios para su funcionamiento:

```
class Renderer implements GLSurfaceView.Renderer {
	private Map map;

	private float posX = 75;
	private float posY = 100;
	private float posZ = 0;


	boolean routeSet = false;
	iDocent miD;

	public Renderer(surfaceView surfaceView){
		map = new Map(surfaceView);
		miD = surfaceView;
	}

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
	  gl.glEnable(GL10.GL_TEXTURE_2D);
	  map.loadTexture(gl, miD);
	  // Configura el fondo a negro
		gl.glClearColor(255.0f, 255.0f, 255.0f, 0.5f);
		// Activa Smooth Shading
		gl.glShadeModel(GL10.GL_SMOOTH);
		// Configura la profundidad del buffer
		gl.glClearDepthf(1.0f);
		// Activa pruebas de profundidad
		gl.glEnable(GL10.GL_DEPTH_TEST);
		// El tipo de pruebas de profundidad a realizar
		gl.glDepthFunc(GL10.GL_LEQUAL);
		// Calculo de perspectiva como nicest.
		gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
	  // Configura el Viewport
		gl.glViewport(0, 0, w, h);
		// selecciona la matriz de proyección
		gl.glMatrixMode(GL10.GL_PROJECTION);
		// resetea la matriz de proyección
		gl.glLoadIdentity();
		// calcula el radio de la ventana
		GLU.gluPerspective(gl, 130.0f, (float) w / (float) h, 0.1f,
				400.0f);
		// selecciona la matriz modelview
		gl.glMatrixMode(GL10.GL_MODELVIEW);
		// resetea la matriz modelview
		gl.glLoadIdentity();
    }

    public void onDrawFrame(GL10 gl) {
	  // limpia la pantalla y el buffer de profundidad
		gl.glClear(GL10.GL_COLOR_BUFFER_BIT
GL10.GL_DEPTH_BUFFER_BIT);
		// remplaza la matriz actual con la matriz identidad
		gl.glLoadIdentity();
		//dibuja el frame con el zoom y traslaciones definidas
		gl.glTranslatef(-(posX+xOffset), -(posY+yOffset), zoom);
		map.Draw(gl);
```

```
            gl.glLoadIdentity();    }
}
```

Una vez dibujado, el grafico dibujado que se mostrará a través del cliente será el siguiente mapa:

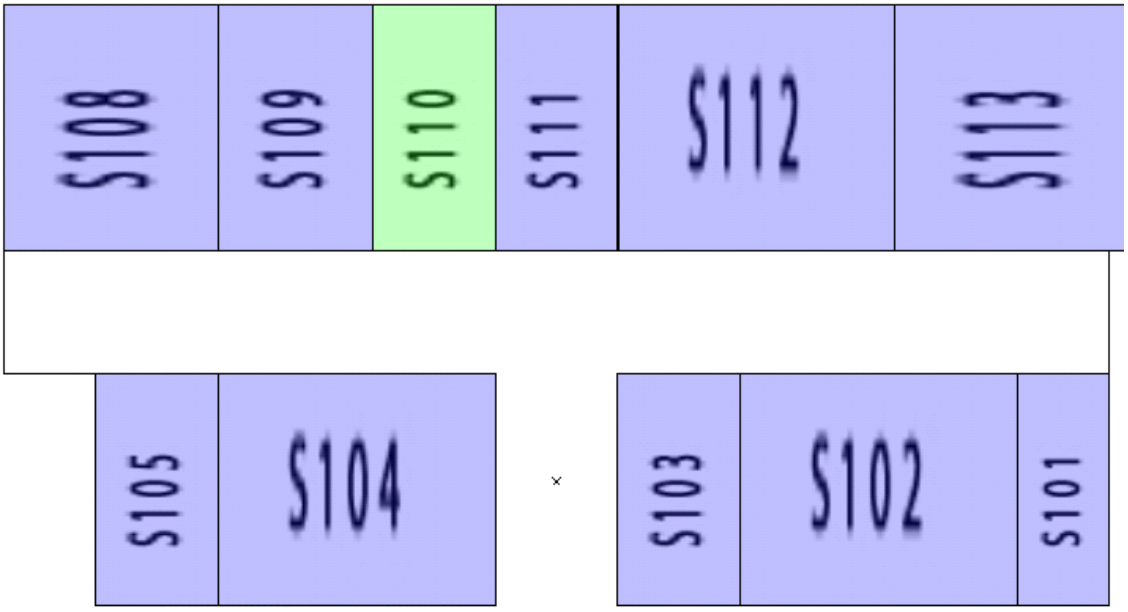


FIGURA 19

6.1.4 Service location

El cliente móvil no solo puede ver el mapa del centro comercial a través de la aplicación sino que también puede ubicarse en él a través de un posicionamiento de coordenadas.

Existen tecnologías móviles que pueden ayudar a localizar los dispositivos, estos son por GPS y Wi-Fi.

6.1.4.1 GPS

Es un sistema de posicionamiento global que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión de hasta centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión.

La precisión del GPS puede verse afectado por muchos factores:

- Retraso de la señal en la ionosfera y la troposfera.
- Señal multirruta, producida por el rebote de la señal en edificios y montañas cercanos.
- Errores de orbitales, donde los datos de la órbita del satélite no son completamente precisos.
- Número de satélites visibles.
- Geometría de los satélites visibles.
- Errores locales en el reloj del GPS.

El GPS que los teléfonos y dispositivos móviles tienen incorporado es el AGPS o GPS Asistido. Fue desarrollado principalmente como un requerimiento del servicio de emergencias estadounidense, el cual requiere la posición de un teléfono móvil en caso de que realice una llamada de emergencia pero ahora se usa ampliamente en varias aplicaciones que incorporan los smartphones.

```
//Obtenemos una referencia al LocationManager
    locManager =
        (LocationManager)getSystemService(Context.LOCATION_SERVICE);

    //Obtenemos la última posición conocida
    Location loc =
        locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

    //Mostramos la última posición conocida
    mostrarPosicion(loc);

    //Nos registramos para recibir actualizaciones de la posición
    locListener = new LocationListener() {
        public void onLocationChanged(Location location) {
            latitud=String.valueOf(loc.getLatitude());
            longitud=String.valueOf(loc.getLongitude());
            precision= String.valueOf(loc.getAccuracy());
        }
    };
```

En este proyecto, el cliente se encuentra dentro de un centro comercial. Por lo tanto la señal del GPS se puede ver atenuada por encontrarse de un edificio incrementando el margen de error en varios metros. Errores de precisión de más de 3 metros son intolerables ya que en todo momento se quiere saber cerca de qué tienda se encuentra el cliente para suministrar esa información a los algoritmos. Una alternativa al uso de GPS es la triangulación de Wi-Fi, que utiliza las coordenadas de los AP (puntos de acceso) para calcular la posición en la que se encuentra actualmente.

6.1.4.2 Wi-Fi

Es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con Wi-Fi, tales como: un ordenador personal, una consola de videojuegos, un smartphone o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso (o hotspot) tiene un alcance de unos 20 metros en interiores y al aire libre una distancia mayor. Pueden cubrir grandes áreas la superposición de múltiples puntos de acceso.

La librería de Android provee un paquete de clases para gestionar las funcionalidades del Wi-Fi en el dispositivo. A través de la API que provee se puede acceder a casi toda la información del dispositivo solicitante, incluyendo la velocidad de los enlaces de red conectados, direcciones IP, el estado de negociación e información de otras redes que están disponibles. También existen ciertas características como escanear, agregar, guardar, terminar o iniciar conexiones Wi-Fi.

Primero que todo, para poder usar las funcionalidades Wi-Fi en una aplicación, se han de definir los permisos correspondientes en el AndroidManifest.XML, un archivo donde se almacenan todos los permisos que tiene la aplicación.

```
<manifest …>
<uses-permission         android:name="android.permission.ACCESS_WIFI_STATE">
</uses-permission>
<uses-permission         android:name="android.permission.CHANGE_WIFI_STATE">
</uses-permission>
<uses-permission      android:name="android.permission.ACCESS_NETWORK_STATE">
</uses-permission>
<uses-permission      android:name="android.permission.CHANGE_NETWORK_STATE">
</uses-permission>
<uses-permission android:name="android.permission.INTERNET">
</uses-permission>
</manifest>
```

Para poder triangular la posición se necesitan las coordenadas de los puntos de acceso del centro comercial y para calcular la distancia a un AP se utiliza la intensidad de señal. El dispositivo lo que hace es cagar las coordenadas de los AP que tiene conocimiento y luego procede a realizar un escaneo de todos los AP que están a su alcance. Si se trata un AP conocido calcula la intensidad de la señal y la almacena para realizar la triangulación según el siguiente algoritmo:

```
public void run() {
            …
//Realiza el escaneo asignando un valor entre 1 y 20 a los AP basándose en la fuerza de la señal y lo guarda en la variable "level"
                  for (ScanResult scan : scans)
                  {
                        int                     level                     =
WifiManager.calculateSignalLevel(scan.level, 20);
// Busca si el AP scan.BSSID es uno de los suyos en tal caso, a la coordenada actual x,y,z del dispositivo le suma "level" veces la coordenada x,y,z del AP respectivamente. Esto se repite para todos los AP encontrados
                        WeightedScan wScan = wsFactory.Create(scan.BSSID);
                        if(wScan != null)
                        {
                              wScan.SetLevel(level);
                              sumX += level*wScan.GetPos().get(x);
                              sumY += level*wScan.GetPos().get(y);
                              sumZ += level*wScan.GetPos().get(z);
                              count+=level;
                        }
                  }
                 …
}

…
float[] loc = {0,0,0};
      …
//las coordenadas x, y, z son la media de las coordenadas x,y,z de los AP sumadas "level" veces
                  loc[x] += sumX /(float)count;
                  loc[y] += sumY /(float)count;
                  loc[z] += sumZ /(float)count;
      …
       try{
//Se envía la nueva posición al servidor
            URL url = new URL("http://localhost:8080/pfc_web/posicion");
             URLConnection connection = url.openConnection();
             connection.setDoOutput(true);
             connection.setDoInput(true);
             Posicionamiento p,p1;
             ObjectMapper mapper = new ObjectMapper();
              p=new Posicionamiento(pref.getInt("idcliente", 1));
             p.setCoordX(loc[x]);
        p.setCoordY(loc[y]);
        p.setCoordZ(loc[Z]);
     mapper.writeValue(connection.getOutputStream(), p);
…
```

La siguiente imagen muestra un mapa de donde están situados los AP's:

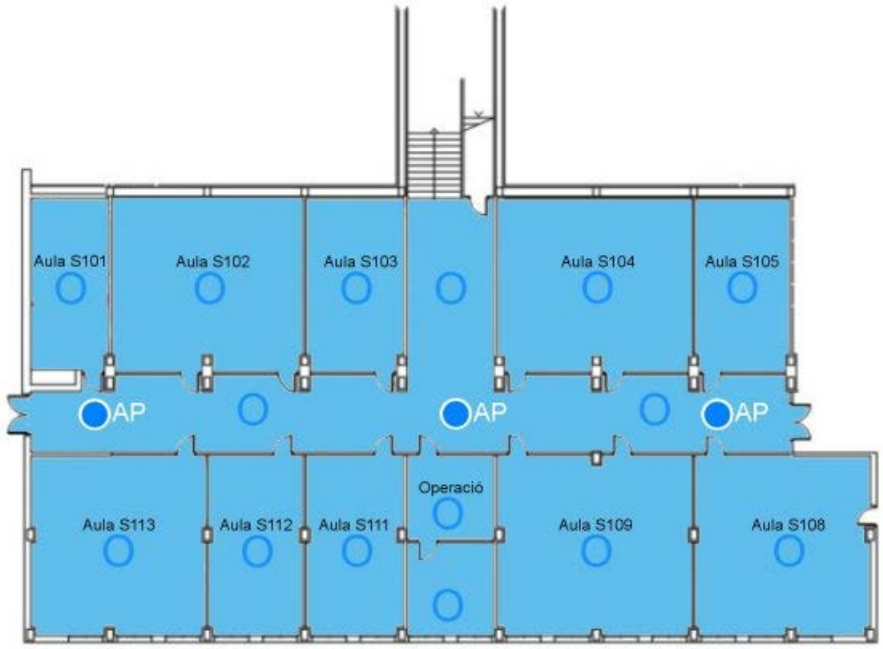


FIGURA 20

6.1.5 Notificación

Una de las funcionalidad que ofrece Android es la de poder notificar mensajes al usuario, como por ejemplo mediante cuadros de dialogo o notificaciones a la barra de estado o los llamados Toast.

6.1.5.1 Toast

Es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento). Aunque son personalizables, aparecen por defecto en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido. Por sus propias características, este tipo de notificaciones son ideales para mostrar mensajes rápidos y sencillos al usuario, pero por el contrario, al no requerir confirmación por su parte, no pueden utilizarse para hacer notificaciones demasiado importantes.

FIGURA 21

6.1.5.2 Cuadros de diálogos

Un cuadro de dialogo es usualmente una pequeña ventana que aparece en frente de la Activity actual. La Activity subyacente pierde el foco y el dialogo acepta toda la interacción del usuario. Los cuadros de diálogos se usan normalmente para notificaciones que interrumpen al usuario y realizan pequeñas tareas que se relacionan directamente con la aplicación en progreso (como una barra de progreso o inicio de sesión)

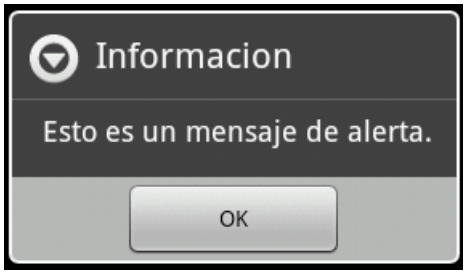


FIGURA 22

6.1.5.3 Barra de estado

Una notificación de barra de estado añade un icono a la barra de estado del sistema (con un pequeño mensaje de texto opcional) y un mensaje de notificación en la ventana de notificaciones. Cuando el usuario selecciona la notificación, Android lanza un Intent definido por la notificación. Entre las opciones que presenta este tipo de notificaciones esta el hecho que se pueden enviar alertas al usuario con sonido, vibración y el led de notificaciones encendido.

Las notificaciones de barra de estado son mayormente usadas para casos donde algún servicio en segundo plano necesita alertar al usuario de un evento que necesita su respuesta.

La aplicación desarrollada en este proyecto consulta con el servicio del servidor cada cierto tiempo si existe alguna publicidad disponible para el cliente en ese momento. Dado que, en caso de recibir alguna publicidad, se necesita notificar del evento de nueva publicidad disponible al usuario requiriendo una respuesta de su parte pero a la vez evitando lo máximo posible interrumpir su actividad con el dispositivo, la notificación por barra de estado es la mejor opción.

Para lograr que la aplicación interrogue al servidor para saber si existe alguna publicidad disponible para el cliente cada cierto tiempo se hará uso de un temporizador. Este estará programado para que cada cierto periodo de tiempo realice la consulta y si detecta que se ha asignado una nueva publicidad al cliente, lanza el evento de notificaciones.

Lo primero que se hace es crear un servicio que, ejecutado cada cierto tiempo, determinará si debe o no lanzarse una notificación. Este servicio extiende la clase Service e implementa runnable, de forma que esta se pueda ejecutar en segundo plano. En el método run() se realizará la consulta enviándole la información del cliente del cual quiere saber si existe alguna publicidad disponible.

```
public class EnvioPublicidad {
//Clase que donde el dispositivo recibirá la información de las nuevas publicidades disponibles
	private boolean grupal;
	private int idcliente;
	private Vector<String> titulo;
	private Vector<String> subtitulo;
	private Vector<String> imagen;
	private Vector<Integer> idpublicidad;
	private Vector<Integer> idtienda;

	public EnvioPublicidad()
	{
	}
	…
}
```

```
URL url = new URL("http://192.168.1.35:8080/pfc_web/selector”)
...

EnvioPublicidad ep=new EnvioPublicidad(pref.getInt("idcliente", 1));
ep.setGrupal(false);
ObjectMapper mapper = new ObjectMapper();
mapper.writeValue(connection.getOutputStream(), ep);
ep1 = mapper.readValue(connection.getInputStream(), EnvioPublicidad.class);

if (ep1.getIdpublicidad()!=null)
{
   handler.sendEmptyMessage(1);
}
```

Si la respuesta del servidor no es “null” se llama la función que generará la notificación con la información recibida.

```
//Prepara la actividad que se abrirá cuando el usuario pulse la notificacion
       Bundle b=new Bundle();
         Intent intentNot = new Intent(context, Verpublicidad.class);
         b.putInt("idpublicidad", ep1.getIdpublicidad().get(0).intValue());
              b.putInt("idtienda", ep1.getIdtienda().get(0).intValue());
       b.putString("titulo", ep1.getTitulo().get(0));
       b.putString("subtitulo", ep1.getSubtitulo().get(0));
       b.putInt("imagen", R.drawable.oferta);


       intentNot.putExtras(b);
    //Prepara la notificacion
        Notification notification = new Notification(R.drawable.ic_launcher,
ep1.getTitulo().get(0), System.currentTimeMillis());
notification.setLatestEventInfo(context, "Centro Comercial IAdvert", "Existe
una promoción interesante",
PendingIntent.getActivity(context,              0,              intentNot,
PendingIntent.FLAG_CANCEL_CURRENT));
    //Le añade sonido
        notification.defaults |= Notification.DEFAULT_SOUND;
    //Le añade vibración
        notification.defaults |= Notification.DEFAULT_VIBRATE;

    //Le añade luz mediante LED
        notification.defaults |= Notification.DEFAULT_LIGHTS;

    //La notificación se detendrá cuando el usuario pulse en ella
        notification.flags = Notification.FLAG_AUTO_CANCEL;
```

Ya hemos implementado en el paso anterior el servicio que llama a las notificaciones cuando lo considera oportuno, pero no se esta ejecutando en ningún sitio. Para ello, en la primera Activity que se ejecuta al iniciar la aplicación se iniciará el servicio si estuviese desactivado.

```
…
private static PendingIntent pendingIntent;
AlarmManager am;
…

am = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

if (pendingIntent == null)
{
//La alarma está desactivada, la activamos
setRepeatingAlarm();
}

…
//Programamos la alarma para que consulte al servidor cada 15 segundos
public void setRepeatingAlarm() {
         Intent intent = new Intent(this, alarma.class);
         pendingIntent = PendingIntent.getBroadcast(this, 0,
           intent, PendingIntent.FLAG_CANCEL_CURRENT);
         am.setRepeating(AlarmManager.RTC_WAKEUP, System.currentTimeMillis(),
           (15 * 1000), pendingIntent);
       }
```

Una vez implementada el servicio de notificación, el resultado obtenido es como el siguiente. En la parte superior se muestra icono avisando de la existencia de una nueva notificación y accediendo a la barra de estado, el cliente puede obtener más información sobre que tipo de publicidad se trata y verla si así lo desea.

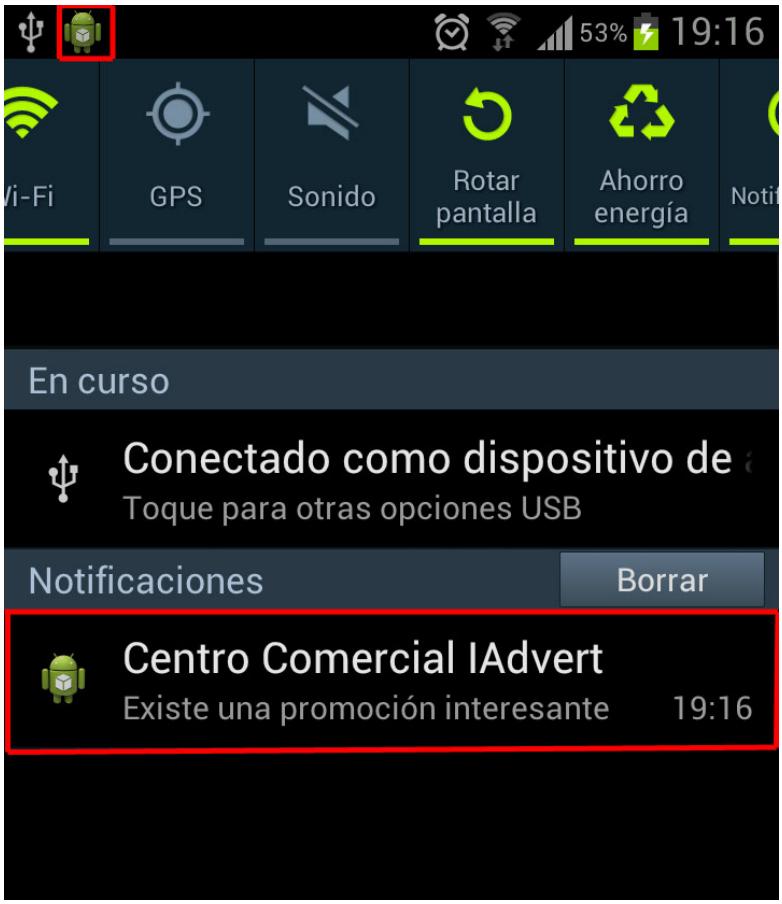


FIGURA 23

6.1.6 Persistencia de datos

Android provee una serie de opciones para mantener la persistencia de datos en una aplicación. La solución que se escoja depende mucho en las necesidades específicas que uno tenga, por ejemplo, si los datos deben ser privados a la aplicación o si puede ser accesible a otras aplicaciones (y el usuario) y cuanto espacio requieren estos datos. Las opciones de almacenamiento de datos son las siguientes:

- **Shared Preferences**: la clase SharedPreferences provee un framework general que permite guardar y recuperar los pares de datos clave-valor persistentes. Estos datos persistirán durante toda la sesión del usuario incluso si la aplicación es cerrada.

- **Almacenamiento interno**: se puede guardar ficheros directamente en el almacenamiento interno del dispositivo. Por defecto, los ficheros guardados en el almacenamiento interno son privados a la aplicación que los crea y otras aplicaciones no pueden acceder a ellos (ni el usuario). Cuando el usuario desinstala la aplicación, estos ficheros son eliminados.

- **Almacenamiento externo**: todos los dispositivos compatibles con Android soportan el almacenamiento externo para guardar ficheros. Estos pueden ser medios de almacenamiento removibles, como la tarjeta SD o uno interno, no removible. Los ficheros guardados como almacenamiento externo pueden ser leídos por cualquiera y pueden ser modificados por el usuario cuando este activa la conexión USB para transferir ficheros a un ordenador.

- **Base de datos SQLite**: Android provee soporte completo para las bases de datos SQLite. Cualquier base de datos que se cree puede ser accesible por nombre en cualquier clase de la aplicación, pero no fuera de ella.

- **Conexión de red**: se puede usar la red (cuando este disponible) para guardar y recuperar datos de un servicio web propio.

De las opciones que ofrece Android, se ha descartado el almacenamiento a través de una conexión de red por el motivo de que cada vez que el cliente quiera hacer uso de la aplicación, este estaría obligado a esta siempre conectado a internet. También se descarta tanto el almacenamiento interno como el externo por la baja seguridad que estos ofrecen y que pueden ser corruptos por terceros (incluso el mismo usuario).

Las opciones más comunes son usar una base de datos con SQLite o Shared Preferences. Para nuestro caso, como lo que se quiere es almacenar información del usuario y de presentación y que estas sean accesibles por toda la aplicación se ha decidido optar por Shared Preferences. El motivo es que Android proporciona este método específicamente diseñado para administrar estos tipos de datos que se quieren guardar y por tanto seria adecuado seguir esta convención.

Shared Preferences usa un fichero XML para mantener los datos persistentes. Para poder acceder estos datos primero se tiene que especificar la Shared Preferences que se quiere usar mediante PreferenceManager y usar los métodos que este ofrece para guardar o recuperar los valores que queremos. La siguiente tabla muestra como funciona este proceso.

```
public class PantallaOpciones extends PreferenceActivity{
…
SharedPreferences pref=PreferenceManager.getDefaultSharedPreferences(
                                        PantallaOpciones.this);

//obtiene el string idcliente y lo guarda en una variable
String idcliente=pref.getString("idcliente", "");

…
// se usa  la interface Editor para guardar un dato con la clave "idcliente"
SharedPreferences.Editor editor = pref.edit();
editor.putInt("idcliente", 1);
editor.commit();
…
```

No necesariamente se debe usar Shared Preferences directamente para guardar las preferencias del usuario. Por otro lado, Android ofrece una clase que provee un framework Activity para crear preferencias del usuario, los cuales son persistentes (usando Shared Preferences). Esta clase es PreferenceActivity.

PreferenceActivity ofrece una interfaz común donde el usuario puede fijar sus preferencias de una forma más cómoda y evitando que el programador tenga que añadir, recuperar o modificar los valores "manualmente" usando directamente SharedPreferences. Una de las ventajas de PreferenceActivity es que permite definir el conjunto de opciones mediante un fichero XML y Android crea por nosotros la pantalla correspondiente para permitir modificarlas a su antojo. El siguiente es un ejemplo de un fichero XML de PreferenceActivity.

```
<?xml …>
<PreferenceScreen … >
    <PreferenceCategory android:title="Opciones de publicidad">
        <ListPreference
            android:key="maxpublidiaria"
            android:title="Máximo publicidad diaria"
            android:summary="Indique el número máximo de promociones
publicitarias diarias"
            android:dialogTitle="Indicar máximo"
            android:entries="@array/maxpublicidad"
            android:entryValues="@array/maxpublicidad" />

        <CheckBoxPreference
                android:key="modopublicidad"
                android:title="Activar envio grupal"
                android:summary="¿Desea recibir la publicidad en forma
grupal?" />
    </PreferenceCategory>
    <PreferenceCategory android:title="Configuración">
        <com.eliot.cliente.ListPreferenceMultiSelect
            android:key="etiquetas"
            android:title="Etiquetas"
            android:summary="seleccione las etiquetas a las cuales quiera
suscribirse"
            android:dialogTitle="Indicar etiquetas"
            android:entries="@array/etiquetas"
            android:entryValues="@array/codigoetiquetas" />
    </PreferenceCategory>
</PreferenceScreen>
```

El resultado de esta PreferenceActivity es el siguiente.

FIGURA 24

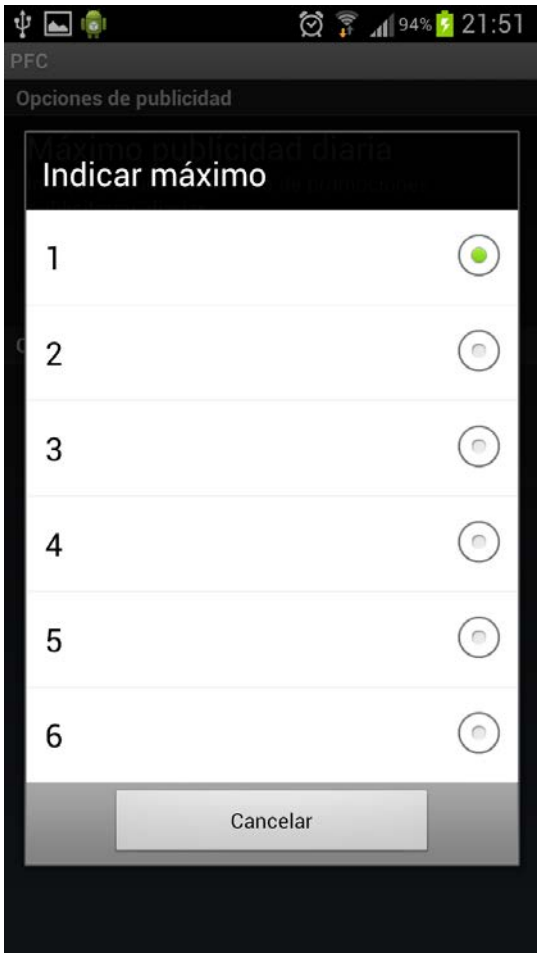


FIGURA 25

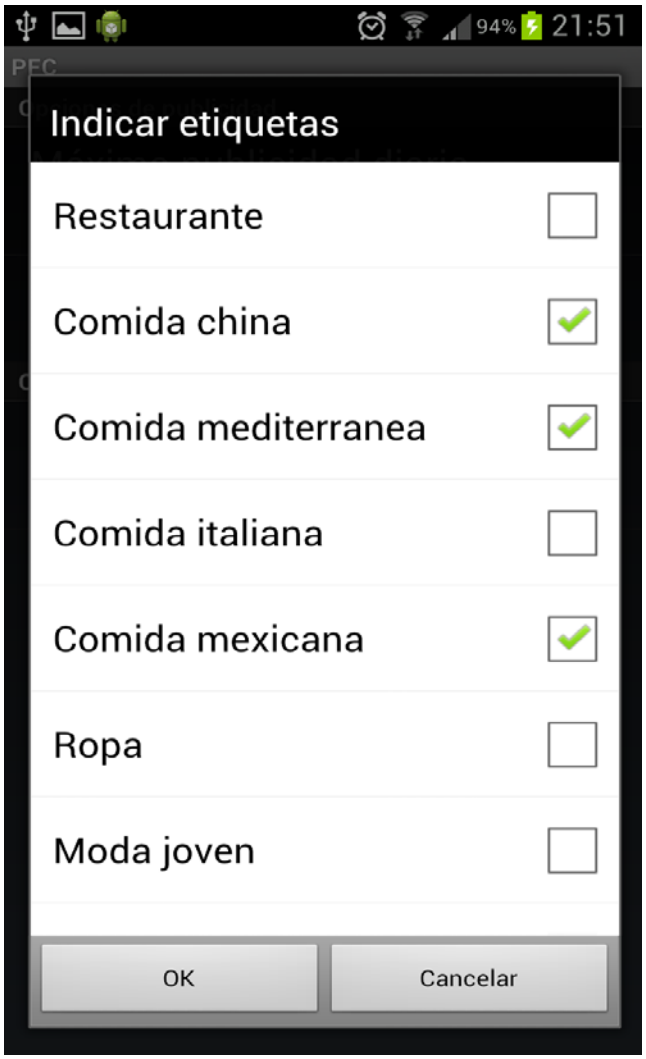


FIGURA 26

6.1.7 Publicidad y código

Cuando el cliente móvil recibe alguna notificación de publicidad, este tiene la posibilidad de visualizarla o no, como se explico en el apartado *Notificación*. Si decide ver la publicidad se abrirá un nuevo activity donde se le muestre en más detalle de que se trata la publicidad y la tienda que la emite.

FIGURA 27

Además, cada publicidad viene con la opción de solicitar el código promocional de la misma. El uso de un código promocional es conveniente para tener un registro de qué publicidades fueron usadas por qué clientes. Otra ventaja de hacer uso de códigos es que de esta manera el sistema podrá retroalimentarse con esta información ayudando al algoritmo de perfil a generar un conocimiento más preciso de los clientes.

En cuanto a la elección del código, existen diferentes formas de generarlos, la forma más simple es creando un código numérico. El cliente nombraría el número de código a la tienda y esta validaría el código con el servicio externo nuestro servidor. A pesar de que esta implementación no es costosa, se puede hacer uso de las nuevas tecnologías que se ofrecen para dispositivos móviles. Esta tecnología es el código QR (Quick Response).

6.1.7.1 Código QR

El código QR es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensionales. Se caracteriza por los tres cuadrados que se encuentran en las esquinas y que permiten detectar la posición del código al lector.

Estos códigos pueden leerse desde PC, smartphones o Tablet mediante dispositivos de captura de imagen, como puede ser un escáner o cámara de fotos, programas que lean los datos QR y una conexión a internet para las direcciones web.

Dado el gran uso de estos códigos, si a los TPV (terminales de Punto de Venta) de las tiendas se les incluyera una cámara con el programa capaz de leer el código (para los de presupuesto bajo) o un dispositivo capaz de escanear el código, por ejemplo un móvil (para los de presupuesto mas alto) el sistema de lectura de códigos podría funcionar correctamente. La ventaja para las tiendas también es alta, ya que estos pueden llegar a saber que tan éxito (porcentaje de uso) esta teniendo su publicidad.

El contenido del código QR es una dirección URL que tiene como parámetro el código único generado llamando al servicio de código y que las tiendas al escanearlo acceden al mismo servicio a través de la URL proporcionada para que el código sea validado. En la siguiente tabla se puede ver la implementación de la generación del código.

```
URL url = new URL("http://192.168.1.35:8080/pfc_web/codigo");
…
//se quiere generar el código para “idcliente” y la publicidad “idpublicidad”
Codigo c=new Codigo(pref.getInt("idcliente", 1),idpublicidad,idtienda);
c.setGenerar(true);
ObjectMapper mapper = new ObjectMapper();
mapper.writeValue(connection.getOutputStream(), c);

//se recibe la respuesta con el código generado
Codigo c1 = mapper.readValue(connection.getInputStream(), Codigo.class);

//genera la imagen del código QR a través de un servicio remoto
imageHttpAddress="https://chart.googleapis.com/chart?chs=150x150&cht=qr&chl=h
ttp://            192.168.1.35:8080/pfc_web/codigo/"+pref.getInt("idcliente",
1)+"/"+idpublicidad+"/"+idtienda+"/"+c1.getCodigo();

URL imageUrl= new URL(imageHttpAddress);
…
//Carga la imagen y la muestra por pantalla
loadedImage = BitmapFactory.decodeStream(conn.getInputStream());
imgvercodigo.setImageBitmap(loadedImage);
```

La siguiente imagen muestra el resultado que se obtiene al hacer clic en “ver código promoción”

FIGURA 28

6.2 JSON

JSON ((JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON.

Dado que el consumo de datos representa un coste para los usuarios móviles se quiere reducir lo máximo el tiempo de respuesta entre el cliente y el servidor, se ha pensado en JSON como la mejor opción por encima de XML.

El siguiente es un ejemplo de como JSON convierte una clase en un formato más ligero para el intercambio de datos:

```
public class Estadisticas implements Serializable{

	int idPublicidad;
	int idCliente;
	int idTienda;
	boolean vista;
	boolean usada;
	boolean eliminada;




	public Estadisticas(){};
	public	Estadisticas(int	idcliente,int	idpublicidad,	int
idtienda,boolean vista,boolean usada,boolean eliminada)
	{
		idCliente=idcliente;
		idPublicidad=idpublicidad;
		idTienda=idtienda;
		this.vista=vista;
		this.usada=usada;
		this.eliminada=eliminada;
	}
	…
}
```

Luego de crear la clase Estadistica, la podemos instanciar de la siguiente forma:

```
Estadisticas e= new Estadisticas(1,2,3,true,false,false);
```

Ahora para poder transformar esta instancia en JSON y enviarlo al servidor hacemos uso de la librería Jackson (FasterXML, 2012), la cual permite hacer la transformación de formato y enviarlo directamente al servidor especificado:

```
URL url = new URL("http://localhost:8080/pfc_web/estadisticas
URLConnection connection = url.openConnection();
connection.setDoOutput(true);
connection.setDoInput(true);
ObjectMapper	mapper	=	new	ObjectMapper();
mapper.writeValue(connection.getOutputStream(), e);
```

Si hacemos un System.out.println para que nos muestre el contenido de la salida de datos, obtenemos lo siguiente:

{"idcliente":1,"idpublicidad:2,"idtienda":3, "vista":true, "usada":false, "eliminada":false}

También en el lado del cliente como del servidor cuando se espera por respuesta un objeto en formato JSON, la librería Jackson también ofrece un método para poder realizar la traducción.

```
Estadisticas e = mapper.readValue(req.getInputStream(), Estadisticas.class);
```

6.3 Servidor

6.3.1 Servicios web

Un servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los web para intercambiar datos en redes de ordenadores como internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Una buena razón por la que los servicios web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día mas utilizada. Entre los estándares mas utilizados para crear servicios web están SOAP y REST.

6.3.1.1 SOAP

SOAP (Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por David Winer, llamado XML-RPC.

SOAP ofrece un framework de mensajería básica en la cual los servicios web se puedan construir. Este protocolo basado en XML consiste de tres partes: un sobre, el cual define qué hay en el mensaje y como procesarlo, un conjunto de reglas de codificación para expresar instancias de tipos de datos, y una conversión para representar llamadas a procedimientos y respuestas.

El protocolo SOAP tiene tres características principales, extensibilidad (la seguridad es una extensión aplicada en el desarrollo), neutralidad (puede ser utilizado sobre cualquier protocolo de transporte como HTTP, SMTP, TCP o JMS) e independencia (permite cualquier modelo de programación).

6.3.1.2 REST

Más que un estándar es una técnica de arquitectura de software para sistemas hipermedia distribuidos como la WWW. Si bien REST se refería originalmente a un conjunto de principios de arquitectura, en la actualidad se usa en el sentido más amplio para describir cualquier interfaz web que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

Un concepto importante en REST es la existencia de recursos (elementos de información), que pueden ser accedidos utilizando un identificador global (un URI – identificador uniforme de recurso). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos (los ficheros que se descargan y se envían).

A diferencia de SOAP, REST no requiere el parseo de ficheros XML ni una cabecera de mensaje a y de un proveedor de servicio, por lo que requiere menos ancho de banda. Debido a estas ventajas que ofrece REST, los servicios que se implementan en el lado del servidor utilizan REST + JSON para la comunicación entre cliente y servidor.

```
public class HomeController {

	…
//servicio que recibe las actividades del cliente móvil con la publicidad
@RequestMapping(value = "/estadisticas", method = RequestMethod.POST)
public @ResponseBody String estadisticas(ServletRequest req) throws
JsonParseException, JsonMappingException, IOException {

Estadisticas e = mapper.readValue(req.getInputStream(), Estadisticas.class);
gu.registrar_actividad(e);
return "1";
	}
// servicio que recibe la posición actual del cliente
@RequestMapping(value = "/posicion", method = RequestMethod.POST)
public @ResponseBody Posicionamiento posicion(ServletRequest req) throws
JsonParseException, JsonMappingException, IOException {
ObjectMapper mapper = new ObjectMapper(); // can reuse, share globally
Posicionamiento p = mapper.readValue(req.getInputStream(),
Posicionamiento.class);
ap.registrar_posicion(p);
return p;
	}
```

```
// servicio que recibe los cambios en el perfil del cliente
@RequestMapping(value = "/perfil", method = RequestMethod.POST)
public   @ResponseBody   Perfiles      perfil(ServletRequest   req)   throws
JsonParseException, JsonMappingException, IOException {
ObjectMapper mapper = new ObjectMapper(); // can reuse, share globally
Perfiles pe = mapper.readValue(req.getInputStream(), Perfiles.class);
ape.registrar_perfil(pe);
return pe;
      }

//servicio que genera los códigos
@RequestMapping(value = "/codigo", method = RequestMethod.POST)
public   @ResponseBody   Codigo      codigo(ServletRequest   req)   throws
JsonParseException, JsonMappingException, IOException {
ObjectMapper mapper = new ObjectMapper(); // can reuse, share globally
Codigo c = mapper.readValue(req.getInputStream(), Codigo.class);

            if(c.isGenerar())
            {
            String   s=ac.generarCodigo(c.getIdCliente(),   c.getIdTienda(),
c.getIdPublicidad());
            c.setCodigo(s);
            }
            return c;
      }

// servicio que valida los codigos
@RequestMapping(value                                                    =
"/codigo/{idcliente}/{idtienda}/{idpublicidad}/{codigo}",      method      =
RequestMethod.POST)
public   @ResponseBody   String      validarcodigo(@PathVariable   String
idcliente,@PathVariable      String      idtienda,@PathVariable      String
idpublicidad,@PathVariable   String   codigo,   Model   model)   throws
JsonParseException, JsonMappingException, IOException {

Codigo   c   =   new   Codigo(new   Integer(idcliente).intValue(),new
Integer(idtienda).intValue(),new Integer(idpublicidad).intValue());
c.setCodigo(codigo);
c.setValidar(true);
if(ac.validarCodigo(c))
      return "Codigo validado";
else
      return "Codigo no validado";
      }

// servicio que envía al cliente las publicidades que se ajustan a su perfil
en un momento determinado

@RequestMapping(value = "/selector", method = RequestMethod.POST)
public  @ResponseBody  EnvioPublicidad   selectorpubli(ServletRequest  req)
throws JsonParseException, JsonMappingException, IOException {
ObjectMapper mapper = new ObjectMapper(); // can reuse, share globally
EnvioPublicidad      ep      =      mapper.readValue(req.getInputStream(),
EnvioPublicidad.class);

EnvioPublicidad ep1=sp.comprobarpublicidad(ep);
return ep1;
      }
}
```

6.3.2 Estadísticas

La función principal del servicio de estadística es recoger todos los datos de actividad que realice el cliente con la publicidad que esta recibiendo. Dado que la información es muy variada y el tipo de datos que se puede recibir depende de lo que haga el cliente. Se utiliza el objeto *Estadísticas* para encapsular toda esta información.

El servicio se encarga de llamar al método que debe registrar esta recopilación en la base de datos y retorna un carácter de confirmación, ya que al ser la comunicación síncrona, el cliente espera una respuesta.

```
public class GestorUso {

…

public void registrar_actividad(Estadisticas e)
{
…
//busca al cliente y publicidad que hace referencia la estadística “e” y los
campos estadísticos que se añadirán
…
//crea una instancia de actividad para el cliente y publicidad
Actividadpublicidad ap=new Actividadpublicidad(api, p,
                          c, e.isUsada(), true,e.isEliminada(),
                          e.isVista());
// guarda si no había alguna actividad o la modifica si ya existía
actividadpublicidadbo.saveorupdate(ap);
}
}
```

6.3.3 Posición

Se ha visto anteriormente que cada cierto periodo de tiempo el cliente móvil envía sus coordenadas de posición al servidor. Este servicio lo que hace es registrar todos estos datos coordenadas recibidas en la base de datos y realizar una serie de comprobaciones para ver si es necesario lanzar la ejecución de algoritmo de publicidad en cumplimiento con el *matching activo*.

Primero que todo, antes de proceder a realizar alguna ejecución que pudiera consumir recursos del servidor, se tiene que realizar comprobar que el cliente se encuentra de hecho dentro del centro comercial. Se pueden recibir coordenadas del cliente porque este puede haber entrado en la aplicación pero si no esta dentro del centro comercial no tiene sentido que sus coordenadas queden registradas.

Por este motivo en la grafica del centro comercial se dibujan circunferencias con una constante R de radio. Cuando se recibe una coordenada se calcula la distancia euclídea de ese punto a los centros de circunferencia. Si la distancia es mayor que R (esta fuera de la circunferencia) quiere decir que esta fuera del centro comercial y se descarta las coordenadas.

Si por el contrario, resulta que esta dentro del centro comercial, se procede a evaluar la coordenada.

La evaluación de las coordenadas se centra principalmente en dos puntos, los que ayudaran al algoritmo de publicidad a realizar el *matching de publicidad*. La primera comprobación que se debe realizar es analizar el punto en que se encuentra y calcular la distancia euclídea a los pares de coordenadas de todas las tiendas. Si la distancia entre estos dos puntos es menor que *r*, quiere decir que esta dentro de la tienda e incrementa el numero de veces que ha estado el cliente en esa tienda así como notifica al sistema para que el algoritmo de publicidad realice el *matching activo*.

El *matching activo* también se activa cuando el cliente cambia de zonas dentro de un centro comercial, este cambio se detecta en el servicio de posición donde se dispone tanto de las coordenadas actuales como de las enviadas en la anterior transmisión. Si tanto las nuevas coordenadas como las antiguas coinciden dentro de una zona no se produce ningún cambio. Por el contrario, si pertenecen a zonas diferentes, el servicio se encarga de notificarlo al sistema.

```
public void evaluar_posicion(int idCliente,double coordX, double coordY,
double coordZ,double coordXant, double coordYant, double coordZant, Date
fecha)
      {
            int id;
            NodoTienda nt;
            …
            //Ubicar la tienda mas cercana
            nt=m.nodoMasCercano(coordX, coordY, coordZ);

           //comprueba si esta dentro la tienda, solo puede estar dentro de
la mas cercana
            if (m.dentroTienda(nt, coordX, coordY, coordZ))
            {
                  //incrementar el numero de visitas a esta tienda +1

                  clientetienda=new ct.findById(idCliente, nt.getIdTienda)
                 clientetienda.setcontador(clientetienda.getcontador()+1);
                  ct.update(clientetienda);
                   //notifica al algoritmo de publicida
                   gp.dentro_tienda(idCliente, nt.getIdTienda);

            }
```

```
      // compara la coordenada actual con la última transmitida
            if (m.cambioZona(coordX, coordY, coordZ, coordXant, coordYant,
coordZant)!=0)
            {
// envía una notificación al algoritmo de publicidad para que ejecute el
matching activo para la situación de cambio de zona
                  gp.cambio_zona(idcliente,coordX,coordY,coordZ);
            }

      }
```

6.3.4 Perfil

Todas las llamadas del cliente móvil al servicio de perfil se pueden realizar por dos motivos, bien porque es la primera vez que el cliente usa la aplicación y el dispositivo se conecta al servicio para pedir la creación de un perfil o bien para modificar los datos que el propio cliente realizar a su voluntad.

Cuando se conecta al servicio por primera vez para que le cree el perfil, lo que hace es sistema es registrar en la base de datos un nuevo usuario y relacionarlo con todas las etiquetas y tiendas del centro comercial. Este proceso es el de crear un perfil por defecto, el cliente no estará suscrito a unas etiquetas ni habrá visto ninguna publicidad que las contenga. Tampoco estará suscrito a alguna tienda ni habrá visitado alguna de ellas en el centro comercial. Por lo tanto la relevancia con la que esta asociada a cada una de ellas es la mínima y todas tendrán la misma.

De esta forma, si bien al principio el envío de publicidades al cliente se puede realizar de forma casi aleatoria, una ves el propio sistema tenga conocimiento de como interactúa con estas y de las modificaciones que realice en su perfil podrá ir refinándose con el tiempo.

Es en el caso que el cliente móvil desee realizar modificaciones directamente sobre sus preferencias o suscribirse a algo cuando el servicio de perfil asume el rol de capturar estos datos.

```
public void registrar_perfil(Perfiles pe)
      {
            …
//si accede para modificar el perfil, remplaza la información personal del
cliente por la nueva
            if(pe.isModificar())
            {
                  …
            }
//modificar perfil si quiere suscribirse a una etiqueta
            if ( pe.getIdTagSuscrita()!=null)
            {
```

```
                …
            }
//modificar perfil si quiere eliminar alguna suscripcion de etiqueta
            if (pe.getIdTagNoSuscrita()!=null)
            {
                …
            }
//modificar perfil si quiere suscribirse a alguna tienda
            if (pe.getIdTiendaSuscrita()!=null)
            {
                …
            }
//modificar perfil si quiere eliminar alguna suscripción de tienda
            if (pe.getIdTiendaNoSuscrita()!=null)
            {
                …
            }
      }
```

Dado que cualquier cambio en el perfil del cliente puede generar un cambio en el conocimiento del mismo, este debe volver a ser generado que siempre este actualizado con los últimos cambios que el cliente haya hecho. Para que la carga de trabajo del servidor no se vea saturada por ejecutar procesos que pueden consumir gran parte del procesador, esta actualización de conocimiento tendrá un temporizador incorporado para que se realice por durante un momento de baja actividad, por ejemplo durante la madrugada, y como hay partes del conocimiento que lo cambian, por ejemplo el historial de relevancias pasadas, solo se ejecutará el algoritmo base del perfil.

```
private void inicializar()
      {
//Las variables que empiezan con $** son una forma abreviada del código original en esa posición para favorecer el entendimiento del mismo.
            …
            valorMaxS=0;
            valorMaxV=0;
            valorMaxI=0;
//busca todas las etiquetas del cliente
            while (itagscliente.hasNext())
            {
                  tagscliente=itagscliente.next();
//multiplica el fator de relevancia que corresponde dependiendo del tipo de etiquetas que sea, las ignoradas son las irrelevantes.
                  if (tagscliente.isSuscrito())
                  {
                  ce.add($idcliente_etiqueta,$contador *Constantes.ALPHA));
                  tipoce.add(($idcliente_etiqueta, "suscrito");
                  if ($contador ()>valorMaxS)
                  valorMaxS=$contador;

                  }

                  else if (tagscliente.isVista())
                  {
```

```
                    ce.add($idcliente_etiqueta,$contador *Constantes.BETA));
                    if ($contador >valorMaxV)
                                valorMaxV=$contador;
                    tipoce.add($idcliente_etiqueta, "vista");
                    }

                    else
                    {
                    ce.add($idcliente_etiqueta, $contador *Constantes.GAMMA));
                    if ($contador ()>valorMaxI)
                    valorMaxI=$contador;
                    tipoce.add($idcliente_etiqueta, "ignorada");
                    }
             }

//Normalización: Sumatorio del vector "ce" es 1 en suscritos, visitadas e
ignoradas;
             itagscliente=listatagscliente.iterator();

             while (itagscliente.hasNext())
             {
             tagscliente=itagscliente.next();
             if (tipoce.get($idcliente_etiqueta).compareTo("suscrito")==0)
            ce.setElementAt(ce.get($idcliente_etiqueta)/valorMaxS),
$idcliente_etiqueta);
             else if (tipoce.get($idcliente_etiqueta).compareTo("vista")==0)
             ce.setElementAt(    (ce.get($idcliente_etiqueta)    /valorMaxV),
$idcliente_etiqueta);
       else if (tipoce.get($idcliente_etiqueta).compareTo("ignorada")==0)
       ce.setElementAt((ce.get($idcliente_etiqueta)/valorMaxI),
$idcliente_etiqueta);
             }
//se repite el procedimiento pero para las tiendas


             valorMaxS=0;
             valorMaxV=0;
             valorMaxI=0;

             while(iclientetienda.hasNext())
             {
                    clientetienda=iclientetienda.next();

                    …
             }

//Normalización: Sumatorio del vector "ct" es 1 en suscritos, visitadas e
ignoradas;
                    iclientetienda=listaclientetienda.iterator();

                    while (iclientetienda.hasNext())
                    {
                           clientetienda=iclientetienda.next();

                        …
                    }
```

```
private void algoritmo_base()
{
//Algoritmo base del perfil
            …
//para publicidad que el cliente ha recibido realiza la obtención de la relevancia real de las etiquetas y tiendas asociadas al cliente según como se especifico en el diseño
            while(iactividadpublicidad.hasNext())
            {
                  actividadpublicidad=iactividadpublicidad.next();

      listatagspublicidad=actividadpublicidad.getPublicidad().getTagspublicidads();
                  itagspublicidad=listatagspublicidad.iterator();
                  while(itagspublicidad.hasNext())
                  {
//recalcula la relevancia para las etiquetas de la publicidad que estén presentes en el perfil del cliente
                        tagspublicidad=itagspublicidad.next();
                        if(actividadpublicidad.isUsada())

      ce.setElementAt((Double)((ce.get($etiqueta_publicidad))*Constantes.ALPHA), $etiqueta_publicidad);
                        else if(actividadpublicidad.isVista())

      ce.setElementAt((Double)((ce.get($etiqueta_publicidad)*Constantes.BETA), $etiqueta_publicidad);
                        else if(actividadpublicidad.isEliminada())

      ce.setElementAt((Double)((ce.get($etiqueta_publicidad)*Constantes.GAMMA), $etiqueta_publicidad);
                  }
//recalcula la relevancia para la tienda de la publicidad que estén presentes en el perfil del cliente

                  if(actividadpublicidad.isUsada())

      ct.setElementAt((Double)((ce.get($tienda_publicidad).doubleValue())*Constantes.ALPHA), $tienda_publicidad);
                  else if(actividadpublicidad.isVista())

      ct.setElementAt((Double)((ce.get($tienda_publicidad).doubleValue())*Constantes.BETA), $tienda_publicidad);
                  else if(actividadpublicidad.isEliminada())
                        ct.setElementAt((Double)((
$tienda_publicidad)*Constantes.GAMMA), $tienda_publicidad);

            }

      }
…
```

6.3.5 Código

Servicio de código se encarga tanto de generar el código de una publicidad para el cliente solicitante como para validarlo cuando una tienda quiere comprobar que es código es correcto y no ha sido usado anteriormente por el cliente.

Anteriormente se ha visto como solicita el cliente el código y cuál es el contenido del mismo. En este apartado lo que se vera es cómo trata el servicio la solicitud, qué método sigue generar un código único en todo momento y como los valida.

Cuando un cliente quiere generar un código para una publicidad, este pasa como información su *id* con el que se identifica a sí mismo ante el sistema y el *id* de la publicidad de cual quiere ver el código. Aprovechando que estos dos identificadores son únicos, el código tendría el siguiente formato.

```
codigo=$idCliente + $idTienda + $idPublicidad;
```

Para este código pueda ser luego validado, se tiene que se hacer una persistencia del mismo en la base de datos. De esta forma cuando el servicio reciba una petición de validación, el sistema buscará en la base de datos primero si hubo un código generado y luego si tanto el código que se tiene almacenado en la base de datos como el que piden validar son el mismo. En tal caso envía un mensaje de confirmación.

6.3.6 Selector

El servicio de selector es consultado periódicamente por el cliente móvil para verificar si existe alguna nueva publicidad disponible que se ajuste al perfil del mismo. A pesar de que el selector únicamente hace una consulta contra la base de datos para ver las publicidades asignadas al cliente, el funcionamiento de como sabe el sistema que publicidad asignarle al cliente responde al algoritmo de publicidad.

Se ha hablado anteriormente de situaciones donde el algoritmo realiza un *matching activo*, cuando se llama a algún método de este tipo de matching, ejecutan una comprobación inmediata del conjunto de publicidades (limitadas por el tipo de matching) que tienen cierta relevancia para el cliente móvil que lo generó.

Por ejemplo, un cliente móvil tiene conjunto de publicidades relevantes asignadas pendientes de enviarse, pero él en su configuración de perfil especifico que solo quiere recibir una publicidad al día. Por lo tanto si está en una zona de restaurantes no tendría sentido enviarle

publicidad de una zona diferente porque su efectividad es menor. Así, el matching activo de cambio de zona seria de gran ayuda para asegurarse de las publicidades que se envían se ajusta al máximo a la actividad que tiene el cliente en esos momentos.

El encargado de poder asignar el conjunto de publicidades más relevantes a los clientes es el matching pasivo. Este hace una búsqueda exhaustiva por publicidad y cliente y los relaciona en el caso que las características de ambos sean compatibles, es decir, que la publicidad tenga una gran relevancia para el cliente.

A continuación se muestra como actúa el algoritmo de matching pasivo.

```
private double[] normalizar_vector (double[] m)
	{
		double var;
		int i;

		//var=valor máximo del vector
		var=0;
		for (i=0;i<m.length;i++)
		{
			if (m[i]>var)
				var=m[i];
		}

		//normaliza vector
		for (i=0;i<m.length;i++)
		{
			m[i]=m[i]/var;
		}
		return m;
	}

private void generamatriz()
	{
// matriz que resultado del producto entre el vector de etiquetas y tiendas
		matriz=new double[ct.size()][ce.size()];

		for (int i=0;i<ct.size();i++)
		{
			for (int j=0;j<ce.size();j++)
			{
				matriz[i][j]=ct.get(i)*ce.get(j);
			}
			matriz[i]=normalizar_vector(matriz[i]);
		}
		…
	}

public void matchingPasivo()
	{
		…

		etiquetas=new Vector<Integer>();
		while(ipublicidad.hasNext())
		{
```

```
                    Publicidad publicidad=ipublicidad.next();

                    settagspublicidad=publicidad.getTagspublicidads();
                    itagspublicidad=settagspublicidad.iterator();
                    // Obtiene todas las etiquetas de la publicidad
                    while(itagspublicidad.hasNext())
                    {
                          tagspublicidad=itagspublicidad.next();
                          etiquetas.add(tagspublicidad.getId().getIdtags());
                    }
                    //Obtiene la tienda que generó la publicidad
                    tienda=publicidad.getId().getIdtienda();

                    //Busca clientes que les encaje esta publicidad
                    iconocimientocliente=vcc.iterator();

                    while (iconocimientocliente.hasNext())
                    {
                          cc=iconocimientocliente.next();
                          matriz=cc.getMatriz();
             //Evalua el cliente para ver si esta publicidad le puede gustar
                          relevancia=0;
                          ietiquetas=etiquetas.iterator();
                          while (ietiquetas.hasNext())
                          {
                                etiq=ietiquetas.next();
                                relevancia+=matriz[tienda][etiq.intValue()];
                          }
                          relevancia=relevancia/etiquetas.size();
                          if (relevancia>=0.6)
                          {
                                //Añadir publicidad al cliente
                                …
                          }

                    }

             }
       }
```

7 Despliegue y pruebas

El sistema desarrollado en este proyecto se despliega en un entorno real para asegurarse que todas las funcionalidades del sistema trabajan sin problemas.

El servidor elegido para realizar las pruebas es el provisto por el laboratorio de investigación de la RDLab de LSI, el cual provee una cuenta y espacio en disco para el despliegue en su servidor "Eagle".

El dispositivo móvil usado en este proyecto es un smartphone que Samsung Galaxy SII, ya que incorpora las últimas tecnologías móviles que existen en el mercado. Además usa el sistema operativo Android, que es un requerimiento ya que el proyecto esta basado en este SO.

Las pruebas de software es un elemento que debe garantizar la calidad del software ya que gracias a ellas comprobaremos que el diseño y la implementación cumplen con la especificación.

En programación, una prueba untaría es una forma de probar el correcto funcionamiento de una modulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

La idea es escribir casos de prueba para cada función no trivial o método en el modulo de forma que cada caso sea independiente del resto.

Se han realizado pruebas de todo tipo para comprobar la correcta ejecución del sistema desde el correcto funcionamiento de la base de datos hasta la comunicación con los clientes. La siguiente es un tipo de prueba que se realiza para comprobar que la capa de negocio puede realizar varios tipos de datos sobre la capa de datos por medio de Hibernate.

```
public class Pruebadatos {

	public static void main(String[] args) {

	ApplicationContext appContext =
	new ClassPathXmlApplicationContext("META-INF/spring/app-context.xml");

	ClienteBo clienteBo = (ClienteBo) appContext.getBean("clienteBo");

	/** insert **/
	Cliente cliente = new Cliente(1,"nombre","contrasena",false);
	clienteBo.save(cliente);

	/** select **/
	cliente= clienteBo.findById(1);

	/** update **/
```

```
        cliente.setMovil("6221548");
        clienteBo.update(cliente);


        /** delete **/
        clienteBo.delete(cliente);

        }

}
```

Para comprobar que los servicios se ejecutaban correctamente, se realizo una versión simple de acceso a los servicios, simulando ser alguna aplicación cliente. Esta prueba también permite comprobar que a parte de haber comunicación entre ellas, pueden entenderse usando el formato JSON.

```
public class ServicioPrueba {

        public static void main(String[] args) throws Exception {

                try {

           URL url = new URL("http://192.168.1.35:8080/pfc_web/estadisticas");
                 URLConnection connection = url.openConnection();
                 connection.setDoOutput(true);
                 connection.setDoInput(true);

                 Estadisticas e=new Estadisticas(2,2,2,true,false,false);

                 ObjectMapper mapper = new ObjectMapper();
                 mapper.writeValue(connection.getOutputStream(), e);
                 System.out.println("ejecutando peticion");

Estadisticas      e1      =      mapper.readValue(connection.getInputStream(),
Estadisticas.class);

System.out.println("Respuesta recibida "+e1.getIdPublicidad());

                } finally {

                }
        }
}
```

8 Costes

En este capítulo se muestra el coste que tendría este proyecto si se desarrollara de forma independiente, teniendo en cuenta la infraestructura hardware, las aplicaciones software y las horas invertidas.

8.1 Costes temporales

El análisis de costes temporales es un estudio del tiempo que finalmente se ha dedicado a cada una de las etapas en el desarrollo del proyecto.

	Planificación	Tiempo real	Desviación
Aprendizaje	70h	88h	18h
Análisis	25h	24h	-1h
Especificación	40h	52h	12h
Diseño	120	128h	8h
Implementación	132	150h	18h
Prototipos	60h	56h	-4h
Pruebas	160h	140h	-20h
Documentación	132	127h	-5h
Total desviación			26h

Debido a que el desarrollo del proyecto incluía el aprendizaje de tecnologías nunca antes usadas como Spring, Hibernate y OpenGL para la creación del mapa, la desviación del aprendizaje se vio considerablemente aumentada ante la necesidad de aprender más cosas para solucionar problemas comunes de implementación.

Respecto a la especificación, la desviación se debió sobretodo a los refinamientos de los casos de uso a medida que se iba hablando con el director del proyecto y delimitar la extensión del mismo, sin embargo el retraso no es muy considerable. Otra fase que produce una gran desviación es en la implementación, que, al estar ligada al uso de tecnologías nunca usadas anteriormente, se retrasaba la misma hasta que haya ampliado el aprendizaje que pudiera solucionar los problemas de implementación.

Gracias a que durante la implementación se probaban los componentes, ya que algunos eran dependientes de otros, en el tiempo que se implementaba se podían realizar al mismo tiempo las pruebas unitarias reduciendo así el número de horas que le correspondían por separado.

8.2 Costes económicos

El análisis económico del proyecto englobará:

- Hardware: la maquinaria utilizada para el desarrollo
- Software: conjunto de herramientas y componentes utilizados para el desarrollo y la documentación.
- Personal: coste temporal del precio de un desarrollador junior

HARDWARE		
Herramienta	Tipo de producto	Coste
i7 Quad Core 1.68Ghz, 8Gb RAM, 100Gb HD	De la empresa	1370€
Samsung Galaxy SII	De la empresa	510€
Total Hardware	1880€	

SOFTWARE		
Herramienta	Tipo de producto	Coste
Eclipse 3.7	Open source	0€
Apache Tomcat 7.0	Open source	0€
Hibernate 3.6.9	Open source	0€
Spring 3.1	Open source	0€
Android 2.1	Open source	0€
MySQL 5	Open source	0€
Microsoft Office	Licencia FIB	0€
Microsoft Project	Licencia MSDSN	0€
Total Software		0€

PERSONAL	**Horas invertidas**
Aprendizaje	88h
Análisis	24h
Especificación	52h
Diseño	128h
Implementación	150h
Prototipos	56h
Pruebas	140h
Documentación	127h

	Horas trabajadas	**Precio/Hora**	**Coste**
Total	765	8.5	6502.5

9 Evaluación final

En un entorno donde en cualquier lugar del mundo uno se puede encontrar con publicidad por todas partes, poder realizar este proyecto me ha permitido analizar como funcionan las técnicas actuales de publicidad y marketing para hacer llegar sus productos a las personas y así buscar una solución que elimine la mala recepción que tienen las personas sobre querer recibir publicidad.

Mientras que por norma general, se intenta que las publicidades lleguen a un gran numero de personas siguiendo la teoría de que mientras mas personas la reciban mayor es la probabilidad de que llegue a alguien que realmente le interese. Sin embargo este proyecto se ha planteado todo lo contrario. Es decir, identificar a aquellas personas que se ha calculado que les puede interesar y hacerles llegar la publicidad.

La solución expuesta en este proyecto usando algoritmos específicos no solo lograría el mismo efecto que otros medios de publicidad sino que también al ser poco intrusivo y evitar el efecto SPAM, las mismas personas tendrían más predisposición a utilizar nuestra aplicación.

Una de las mayores características de este sistema es lograr un sistema de posicionamiento a través de dispositivos móviles, la mayoría de aplicaciones ofrecen el servicio localización de google a nivel de ciudad pero en el caso de este proyecto, al tratarse de posicionar indoor (en el interior del centro comercial), generar un mapa y que el cliente pueda guiarse por medio de él es un valor añadido que le animaría a usar la aplicación con mayor voluntad.

10Mejoras y trabajos futuros

10.1 Mejoras

Uno de los puntos débiles de guardar información en un dispositivo móvil y hacer un intercambio de datos con un servidor es la seguridad. Hasta hace poco tiempo no existían incluso antivirus diseñados específicamente para móviles. La aplicación móvil diseñada en este proyecto, al acceder constantemente a un servidor de nuestro sistema, puede sufrir de un ataque “man in the middle” y por tanto se puede filtrar información personal del usuario.

Una de las mejoras que se plantea es poder usar seguridad SSL en cualquier comunicación con el servidor. De esta forma, al ser todo el contenido cifrado no habrá peligro de filtraciones.

10.2 Trabajos futuros

En el desarrollo de este proyecto se tuvo muy en cuenta la escalabilidad. Al ser la gestión de publicidad un problema que no es propio de un centro comercial, se tiene la idea de poder extender este sistema a lo largo de una ciudad donde todas las tiendas del área urbano pueden participar y enviar sus publicidades a gente que se encuentre en una zona cercana y a pie (se podría diferenciar de aquellos que van en coche por la velocidad a la que pasa de un punto X a uno Y) ya que si fuera en coche las probabilidades de que pare para comprar son bajas.

Otro trabajo futuro que es tiene en cuenta es aplicar este sistema como un remplazo de las tarjetas de fidelización. Actualmente aquellas tiendas que usan tarjetas de fidelización compran el software que viene incorporado con un TPV (terminal de punto de venta).Este software es costoso y no todas las tiendas pueden permitírselo.

El sistema de este proyecto podría ampliarse permitiendo a las tiendas registrar a los usuarios que quieren fidelizarse y cada vez que realizan alguna compra asígnale punto o cupones de descuento. Además, dado que el servidor recoge información de toda la actividad que tiene el cliente móvil con las publicidades y compras, la tienda podría recibir valiosa información sobre el cliente y así ofrecerle promociones personalizadas que beneficiarían ambos, la tienda porque logra retener al cliente y el cliente porque puede aprovechar de las ofertas que se le ofrecen.

11Bibliografía

A Andersson, J. N. (2000). *Wireless Advertising Effectiveness.* Recuperado el 6 de febrero de 2012, de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.198.8147&rep=rep1&type=pdf

B Kim, J. H. (2011). *AdNext: A Visit-Pattern-Aware Mobile Advertising System for Urban Commercial Complexes.* Recuperado el 7 de febrero de 2012, de http://nclab.kaist.ac.kr/papers/Conference/hotmobile11-adnext.pdf

blog, s. (s.f.). *Desarrollo en Android*. Recuperado el 30 de enero de 2012, de http://sgoliver.net/blog/?p=1313

comscore. (23 de febrero de 2012). *Prensa y eventos*. Recuperado el marzo de 2012, de http://www.comscore.com/esl/Press_Events/Press_Releases/2012/2/comScore_Releases_the_2012_Mobile_Future_in_Focus_Report

D Quercia, G. D. (2011). *Mobile phones and outdoor advertising: Measurable advertising.* Recuperado el 6 de febrero de 2012, de http://www.cl.cam.ac.uk/~dq209/publications/quercia11mobile.pdf

Developers, G. (2012, marzo 26). *Infographics*. Retrieved abril 9, 2012, from Getting Started with Infographics: https://developers.google.com/chart/infographics/docs/overview?hl=es-ES

FasterXML. (6 de marzo de 2012). *JacksonHome*. Recuperado el 15 de abril de 2012, de JacksonHome: http://wiki.fasterxml.com/JacksonHome

findemor. (octubre de 2010). *Programar en Android: Notificaciones.* Recuperado el 14 de mayo de 2012, de findemor: http://blog.findemor.es/2011/10/programar-en-android-notificaciones/

García, A. P. (13 de junio de 2007). *Hibernate Tools y la generación de código.* Recuperado el 25 de marzo de 2012, de AdictosAlTrabajo: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateTools

García, A. P. (27 de septiembre de 2010). *Spring + REST + JSON = SOAUI.* Recuperado el 28 de abril de 2012, de AdictosAlTrabajo: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=springRestJson

Gavin King, C. B. (s.f.). *Hibernate.* Recuperado el 25 de marzo de 2012, de Community Documentation: http://www.davidmarco.es/tutoriales/hibernate-reference/

Hristova, N. (2004). *Ad-me: wireless advertising adapted to the user location, device and emotions.* Recuperado el 5 de febrero de 2012, de http://osl-vps-2.ucd.ie/~rem/PRISM-InfoPack/Mobile/HICSS_NO.pdf

IAB. (Marzo de 2012). *Glosario de términos de publicidad y marketing digital.* Recuperado el 18 de Marzo de 2012, de Interactive Advertising Bureau: http://www.iabspain.net

iDOCENT. (n.d.). Retrieved abril 20, 2012, from Indoor Wireless Navigation: http://www.egr.msu.edu/classes/ece480/capstone/spring11/group02/index.html

Phutela, D. (s.f.). *Hibernate vs JDBC.* Recuperado el 25 de marzo de 2012, de mindfiresolutions: http://www.mindfiresolutions.com/mindfire/Java_Hibernate_JDBC.pdf

T Saari, N. R. (2004). *Psychologically targeted persuasive advertising and product information in e-commerce.* Recuperado el 6 de febrero de 2012, de http://wing.comp.nus.edu.sg/downloads/keyphraseCorpus/155/155.pdf

Torrijos, R. L. (s.f.). *Catálogo de Patrones de Diseño J2EE. I.- Capa de Presentación*. Recuperado el 12 de abril de 2012, de Programacion en castellano: http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee__i__capa_de_presentacion_240/4#pagina4

V Toubiana, A. N. (2010). *Adnostic: Privacy preserving targeted advertising.* Recuperado el 5 de febrero de 2012, de http://www.nyu.edu/pages/projects/nissenbaum/papers/adnostic.pdf

Vatanparast, R. (2010). *Pierce the fog of Mobile service and advertising adoption.* Recuperado el 5 de febrero de 2012, de http://lib.tkk.fi/Diss/2010/isbn9789526030159/isbn9789526030159.pdf

W3C. (1997, junio 2). *Proposal for an Open Profiling Standard*. Retrieved febrero 12, 2012, from http://www.w3.org/TR/NOTE-OPS-FrameWork.html

WRAY, J. (2009). *MOBILE ADVERTISING ENGINE.* Recuperado el 5 de febrero de 2012, de http://www2.stetson.edu/mathcs/people/students/research/pdf/2008/jwray/final.pdf